Introduction to Artificial Intelligence

丁尧相
浙江大学

Fall & Winter 2022
Week 12

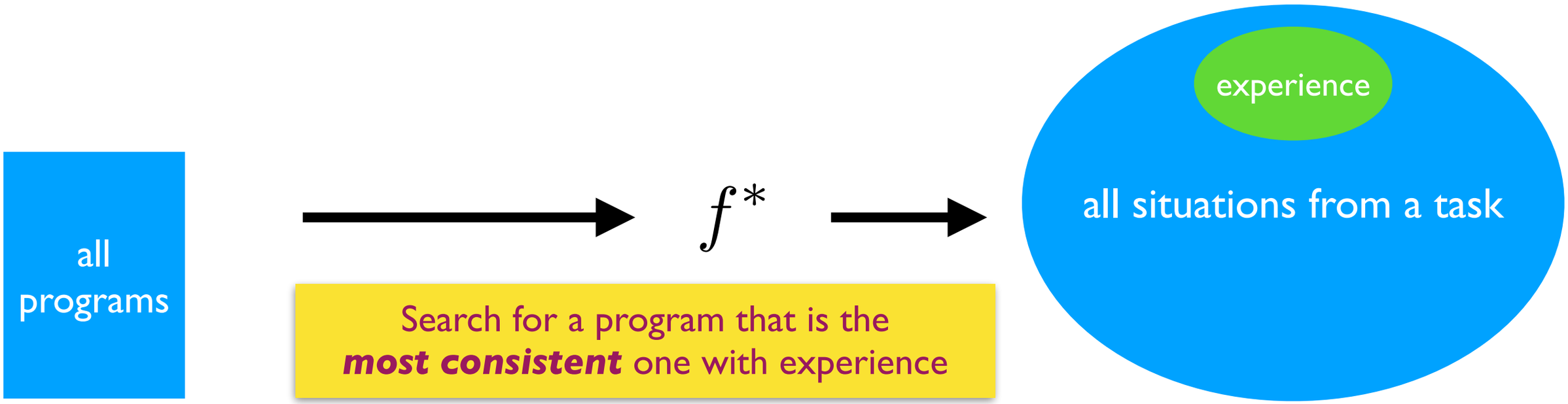
Announcements

- Problem set 4.1 will be released next week.
- Due date of lab project 1 is finalized: next Monday (Dec. 5).
- We have released lab project 2.

Machine Learning: III

- From linear model to neural network
- Feedforward neural network
- Optimization
  - Stochastic gradient descent
  - Back-propagation
- Take-home messages

Machine Learning



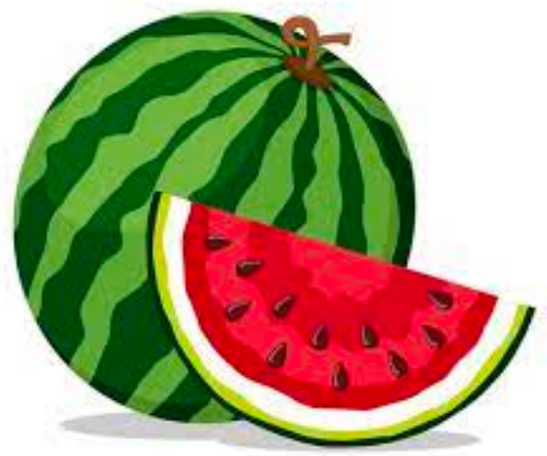
"Machine learning = task + data + objective + algorithm".

— Tom Mitchell

Example: Watermelon Classification

- Build a program to detect whether a watermelon is good or not.
- The program: $f : X \to Y$
  - X : features of watermelon, e.g. color, root type, touch sound.
  - Y : binary variable, indicating good or bad.
- Dataset: instances of (X, Y) pairs:

编号	色泽	根蒂	敲声	好瓜
1	青绿	蜷缩	浊响	是
2	乌黑	蜷缩	浊响	是
3	青绿	硬挺	清脆	否
4	乌黑	稍蜷	沉闷	否

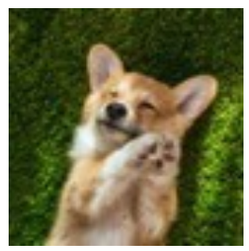


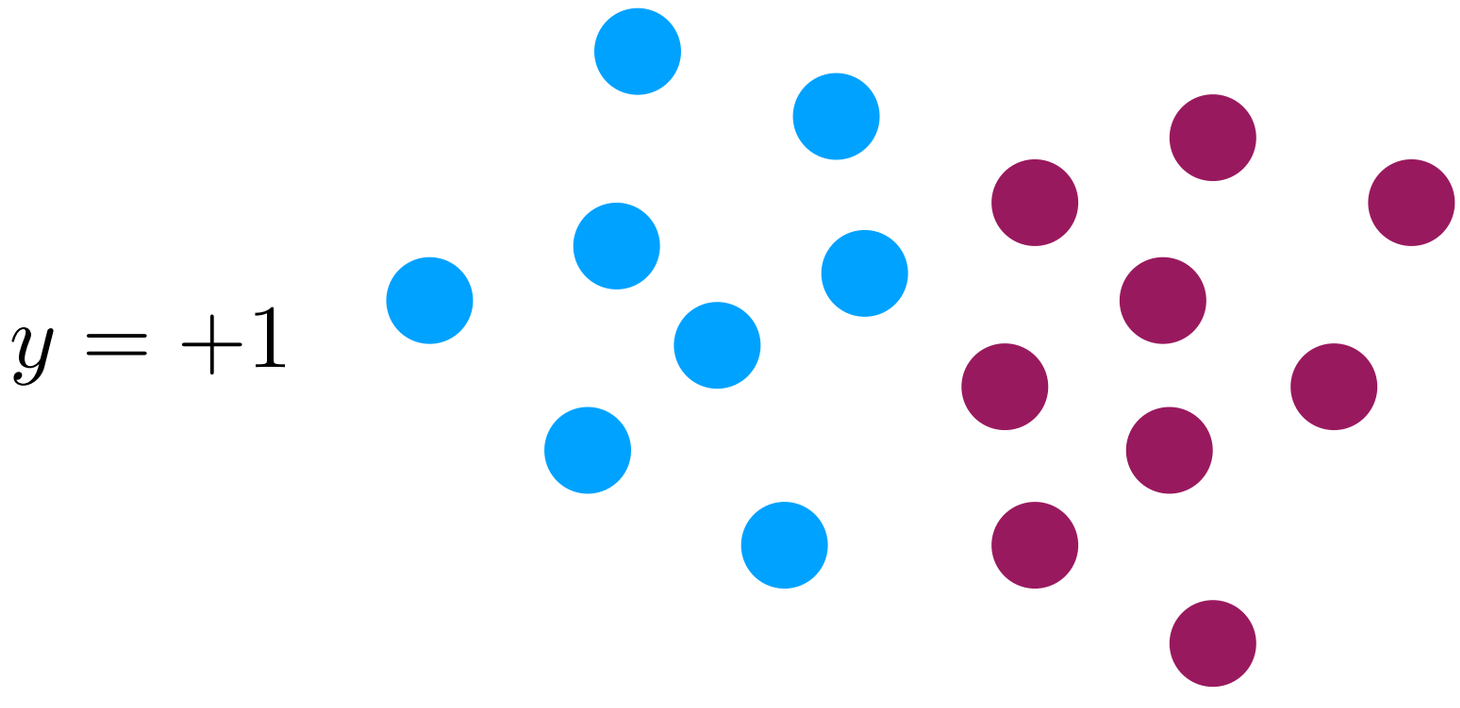
The target is to find the best f^* which has the highest classification accuracy.

Binary Classification

- Task: learn $f : X \to Y$ with $Y = \{-1, 1\}$
- Performance measure: correctness of classification $\mathbb{I}[f(x) = y]$
- Dataset: $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$

vs.

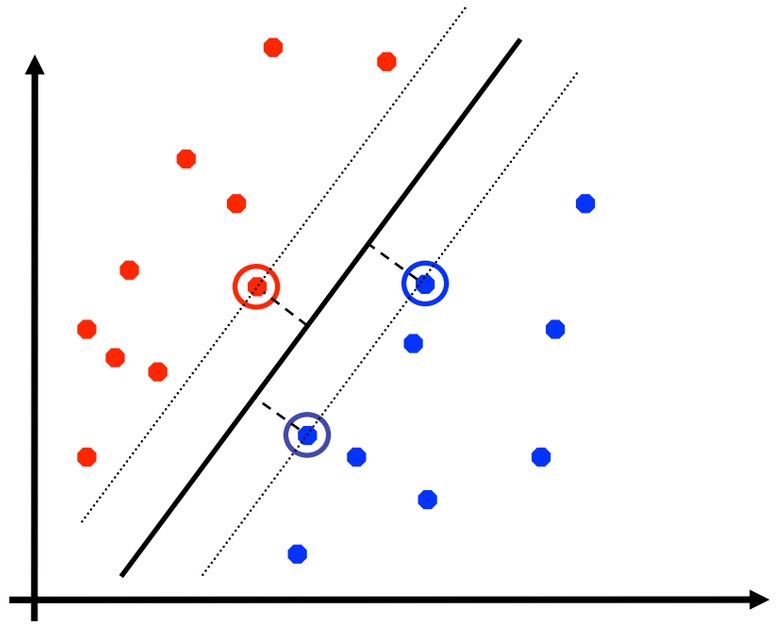




Support Vector Machine (SVM)

- SVMs (Vapnik, 1990's) choose the linear separator with the **largest margin**

Robust to outliers!



V. Vapnik

- Good according to intuition, theory, practice
- SVM became famous when, using images as input, it gave accuracy comparable to neural-network with hand-designed features in a handwriting recognition task

Large-Margin Voting Classifiers

- SVM learns the max-margin linear model: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{b}$
- We can relax the formulation into $H(\mathbf{x}) = \sum_{t=1}^{T} w_t h_t(\mathbf{x})$, in which $h_t(\mathbf{x})$ can be any transformation of $\mathbf{x}$.
- In special, we assume $h_t(\mathbf{x})$ to be some base classifiers. Then $H(\mathbf{x})$ is the voting result of the base classifiers.

We want to combine the power of these base classifiers to form a stronger voting classifier.
The key idea lies also on margin maximization.

AdaBoost

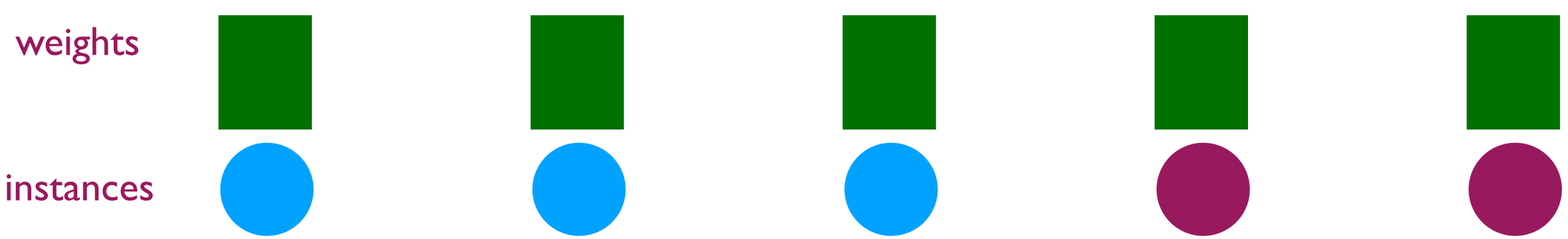
- In each boosting iteration, learn a new base classifier based on instance distribution weights.

- Margin calculation:

$$h_1 : y_1 h_1(\mathbf{x}_1) = 1, y_2 h_1(\mathbf{x}_2) = -1, y_3 h_1(\mathbf{x}_3) = 1, y_4 h_1(\mathbf{x}_4) = 1, y_5 h_1(\mathbf{x}_5) = -1$$

- Set weight: margin large then model weight large, otherwise model weight small: want to make more instances to have larger margins.

- Change data distribution: increase instance weight for small-margin instances.



AdaBoost

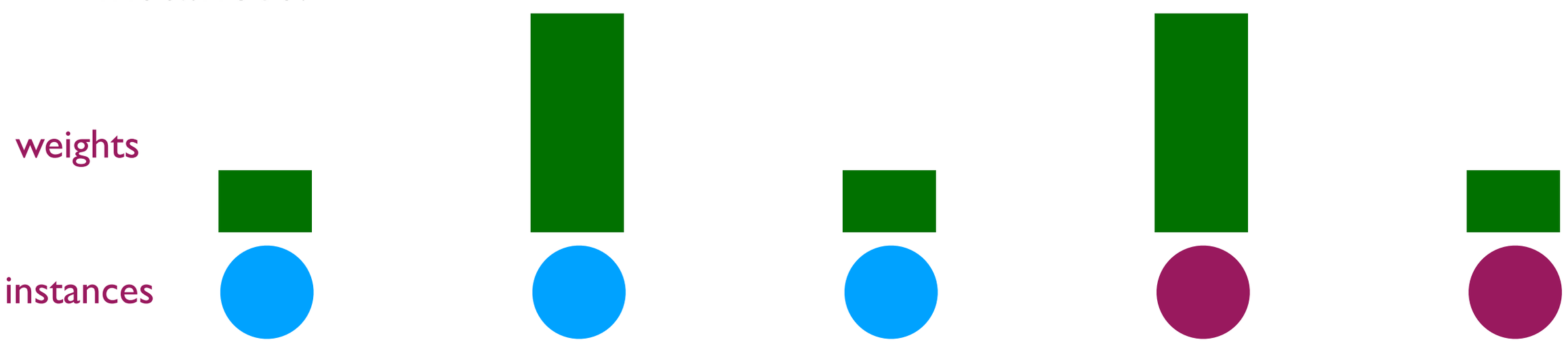
- In each boosting iteration, learn a new base classifier based on instance distribution weights.

- Margin calculation:

$$h_1 : y_1 h_1(\mathbf{x}_1) = 1, y_2 h_1(\mathbf{x}_2) = -1, y_3 h_1(\mathbf{x}_3) = 1, y_4 h_1(\mathbf{x}_4) = 1, y_5 h_1(\mathbf{x}_5) = -1$$

- Set weight: margin large then model weight large, otherwise model weight small: want to make more instances to have larger margins.

- Change data distribution: increase instance weight for small-margin instances.

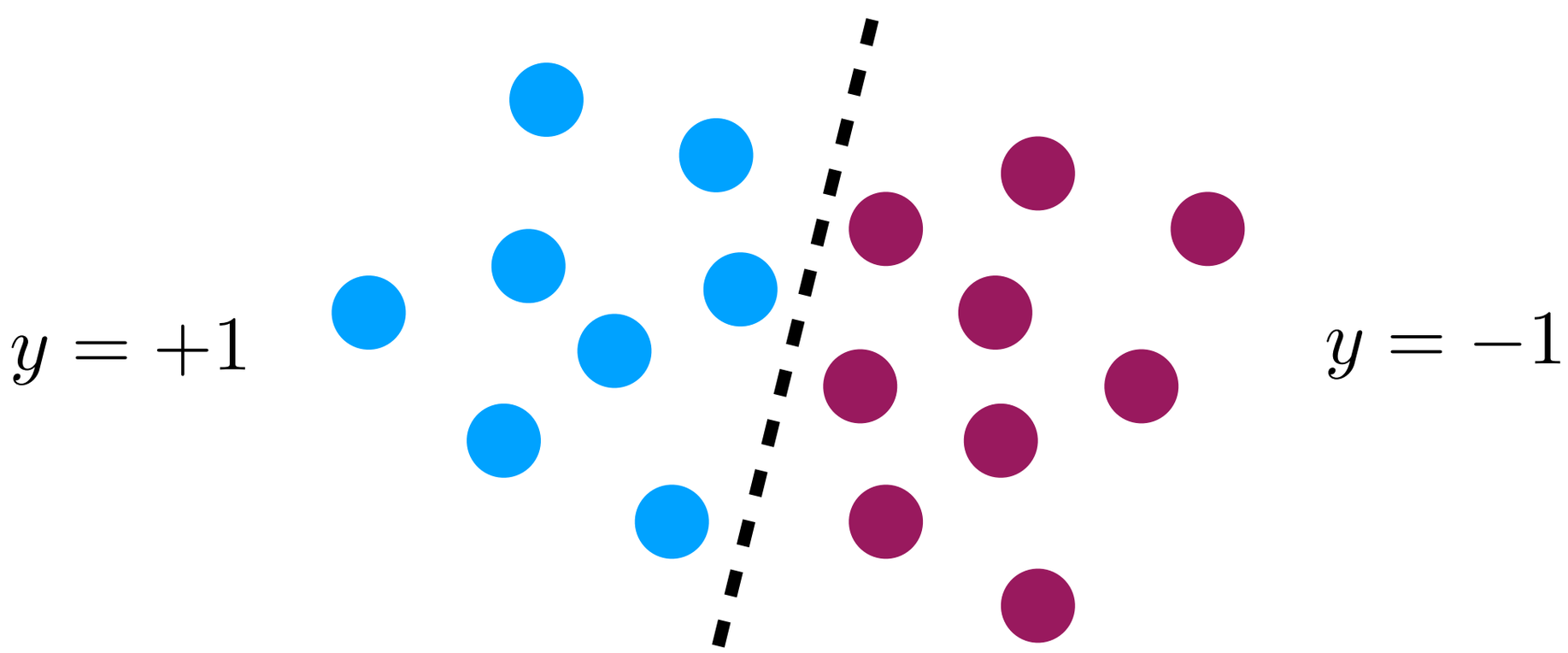


Machine Learning: III

- From linear model to neural network
- Model: feedforward neural network
- Optimization
  - Stochastic gradient descent
    - Back-propagation
- Take-home messages

Linear Model

- Linear model: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + \mathbf{b}$, $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d, \mathbf{b} \in \mathbb{R}$.
- Classification rule: $f(x) > 0$ then predict positive, $f(x) < 0$ negative.
- Basic idea: Find the model performs good under the dataset.
- Observation: $\mathbb{I}[f(x) = y]$ is hard to optimize directly.



Statistical Decision Theory

- Assume that the target function $f : X \to Y$ can be formulated as the conditional distribution $p(y|\mathbf{x})$, i.e. a probabilistic model.

Statistical Decision Theory

- Assume that the target function $f : X \to Y$ can be formulated as the conditional distribution $p(y|\mathbf{x})$, i.e. a probabilistic model.

- For binary classification, the optimal decision rule is

$$f(\mathbf{x}) = \begin{cases} +1, & p(y = +1|\mathbf{x}) \geq 0.5 \\ -1, & p(y = +1|\mathbf{x}) < 0.5. \end{cases}$$

Why? This decision rule has minimum expected error.

Statistical Decision Theory

- Assume that the target function $f : X \to Y$ can be formulated as the conditional distribution $p(y|\mathbf{x})$, i.e. a probabilistic model.

- For binary classification, the optimal decision rule is

$$f(\mathbf{x}) = \begin{cases} +1, & p(y = +1|\mathbf{x}) \geq 0.5 \\ -1, & p(y = +1|\mathbf{x}) < 0.5. \end{cases}$$

Why? This decision rule has minimum expected error.

- Generalization: For multi-class classification, the optimal decision rule is to choose the $\hat{y}$ with the largest $p(\hat{y}|\mathbf{x})$

We should choose suitable models to represent $p(\hat{y}|\mathbf{x})$.

Maximum Likelihood Estimator

- Assume that $p(y|\mathbf{x})$ is represented by a function which has parameter θ: $p(y|\mathbf{x};\theta)$

- In this formulation, machine learning can be treated as the problem of estimating θ given the training data $\mathcal{S}$.

- One of the fundamental tools to solve this problem is to do maximum likelihood estimation:

$$\theta^* = \max_\theta \log p(\mathcal{S}|\theta)$$

$$= \max_\theta \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i;\theta)$$

This is based on the assumption that the data are independent & identically distributed, and the frequentist assumption "all θ are equal before seeing data".

Logistic Regression

- Logistic regression (LR) is a binary classification model.
- LR builds upon the generalized linear model to represent $p(y|\mathbf{x})$:

$$p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

- $p(x) = \frac{1}{1+e^{-x}}$ is called sigmoid function, which is monotonically increasing.

Larger $\mathbf{w}^T\mathbf{x}$ leads to larger $p(y|\mathbf{x})$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \dfrac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\mathbb{I}[y_i = +1] \log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i = -1] \log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \max_{\theta} \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}) \right]$$

$$= \min_{\theta} \sum_{i=1}^{N} \left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}) \right]$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \underbrace{\left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]}_{\text{logistic loss}}$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \underbrace{\left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]}_{\text{logistic loss}} \qquad \underbrace{\max\left(0, 1-y_i(\mathbf{w}^T\mathbf{x}_i+\mathbf{b})\right)}_{\text{hinge loss}}$$

Logistic Loss

- The MLE estimator for logistic regression $p(y=+1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$:

$$\text{MLE}(\theta) = \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{e^{-\mathbf{w}^T\mathbf{x}_i}}{1+e^{-\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\mathbb{I}[y_i=+1]\log(\frac{1}{1+\exp^{-\mathbf{w}^T\mathbf{x}_i}}) + \mathbb{I}[y_i=-1]\log(\frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \max_\theta \sum_{i=1}^{N} \left[\log(\frac{1}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\right]$$

$$= \min_\theta \sum_{i=1}^{N} \left[\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})\right]$$

logistic loss

$$\max\left(0, 1-y_i(\mathbf{w}^T\mathbf{x}_i+\mathbf{b})\right)$$

hinge loss

Logistic regression can also be formulated into loss minimization.
Regularization and kernel method can also be used!

Cross-Entropy Loss

- Logistic loss $\log(1 + e^{-y\mathbf{w}^T\mathbf{x}})$ is designed for linear model in binary classification.

- For general probabilistic model $f(\mathbf{x})$, we can design a similar loss:

$$-\Big[\mathbb{I}[y = +1]log(p) + \mathbb{I}[y = -1]log(1 - p)\Big]$$

p is the abbreviation of $p(x)$

This is called the cross-entropy loss for binary classification.
It is equivalent to logistic regression when using linear model.

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

- The multi-class logistic loss is

$$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \quad \rightarrow \quad -\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

- The multi-class logistic loss is

$$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \quad \rightarrow \quad -\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

- In general, $p(y_c|\mathbf{x})$ can represented by other functions, then the multi-class cross-entropy loss is also

$$-\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

Multi-Class Cross-Entropy Loss

- For multi-class classification with $y \in \{1, 2, ..., C\}$, extend logistic regression model as

$$p(y = +1|\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} \quad \rightarrow \quad p(y_c|\mathbf{x}) = \frac{e^{\mathbf{w}_c^T\mathbf{x}}}{\sum_{i=1}^{C} e^{\mathbf{w}_c^T\mathbf{x}_i}}$$

There is a parameter $\mathbf{w}_c$ for each class.

- The multi-class logistic loss is

$$\log(1 + e^{-y\mathbf{w}^T\mathbf{x}}) \quad \rightarrow \quad -\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

- In general, $p(y_c|\mathbf{x})$ can represented by other functions, then the multi-class cross-entropy loss is also

$$-\sum_{c=1}^{C} \mathbb{I}[y = y_c] \log p(y_c|\mathbf{x})$$

Usually the most common choice for classification with NN.

Optimization: Gradient Descent

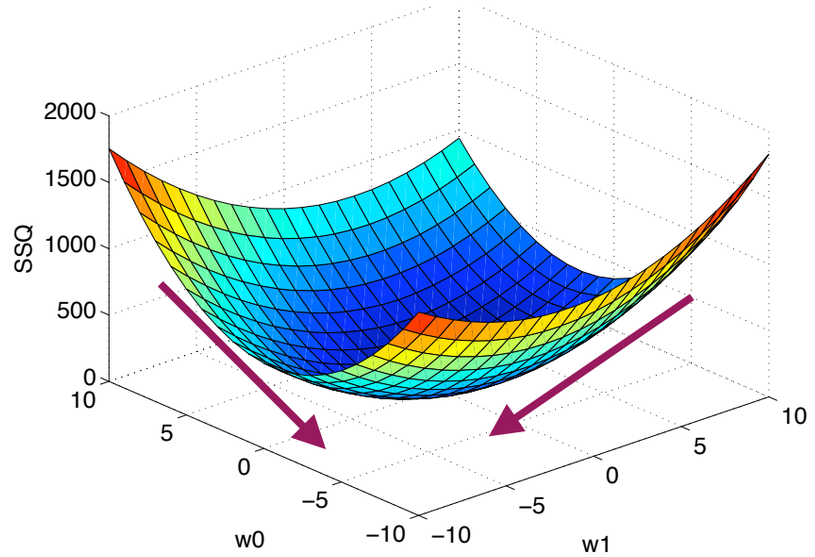
- Logistic loss is a convex function w.r.t. $\mathbf{w}$

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \, \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$



Tracking the gradient direction to find the minimum

Gradient descent is guaranteed to find the global minimum for convex functions.

Linear Classifier for Images

Stretch pixels into column

$$f(x,W) = Wx + b$$

Input image
(2, 2)

0.2	-0.5	0.1	2.0
1.5	1.3	2.1	0.0
0	0.25	0.2	-0.3

W (3, 4)

56
231
24
2

(4,)

+

1.1
3.2
-1.2

b
(3,)

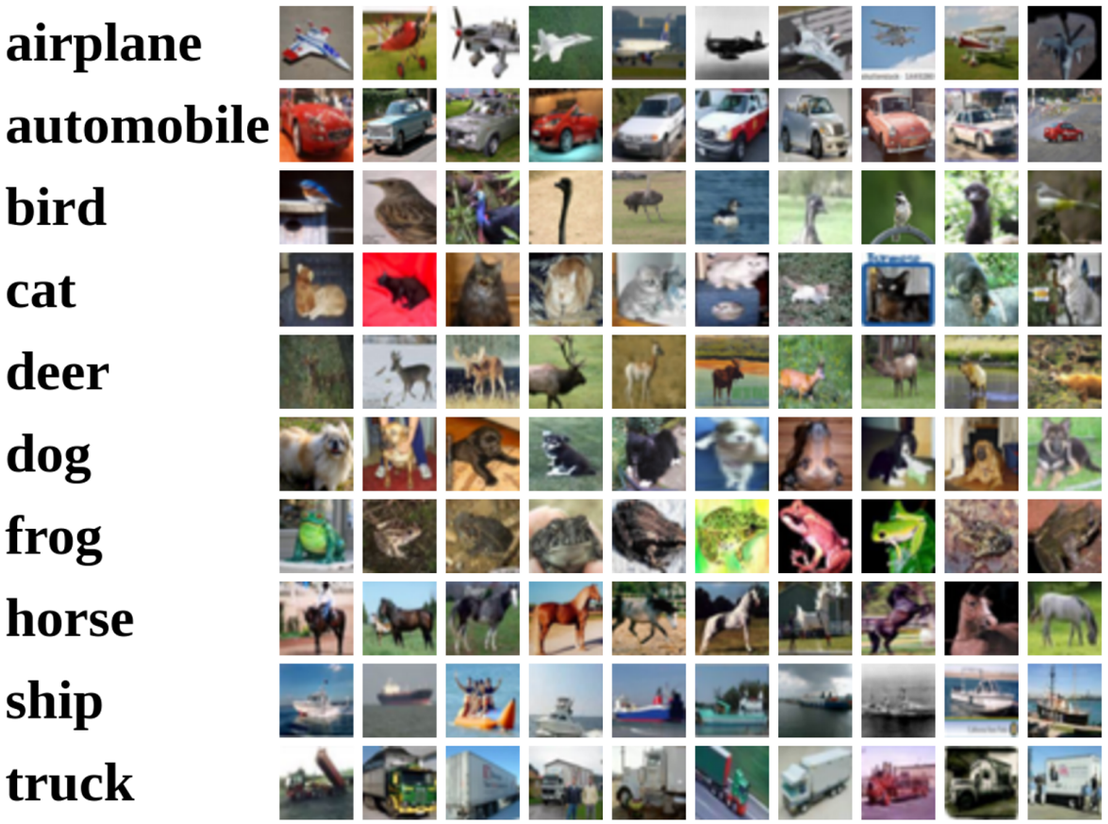
=

-96.8
437.9
61.95

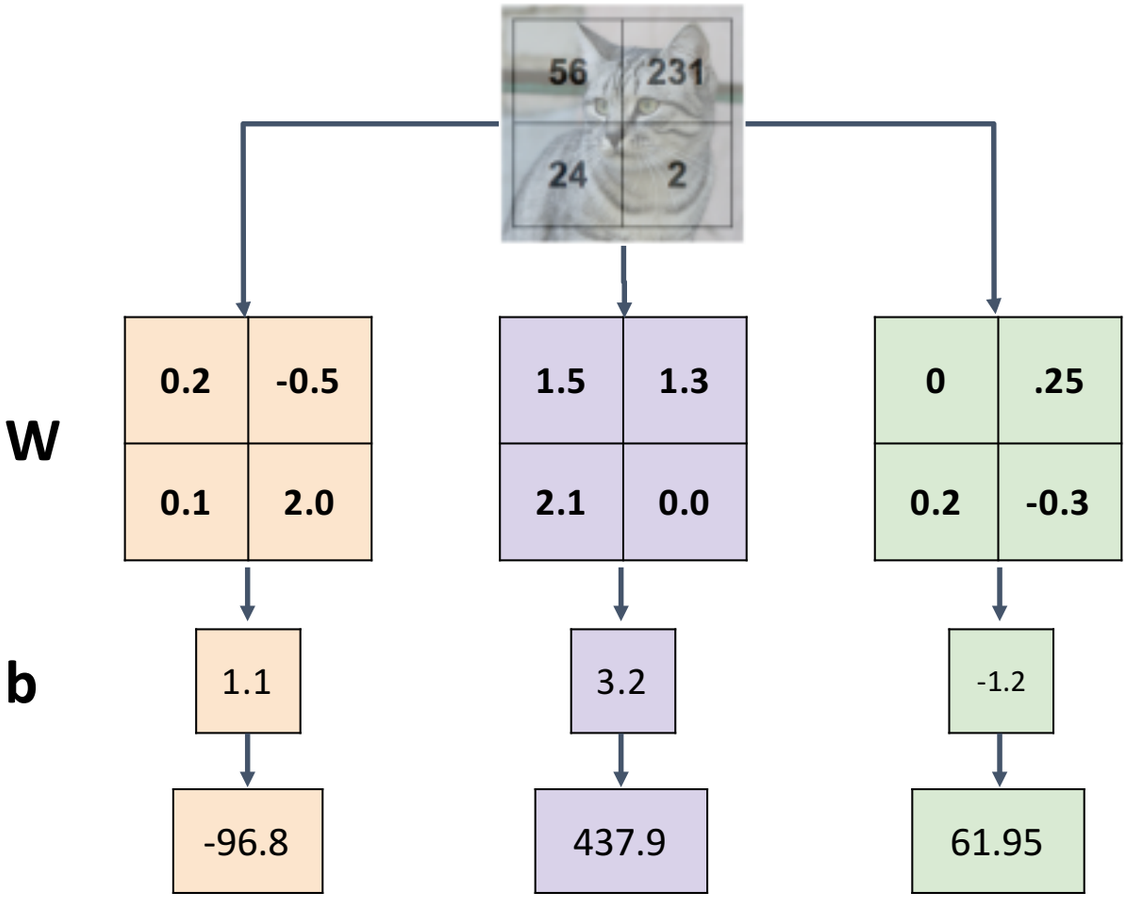
(3,)

Linear Classifier: Weights are Templates

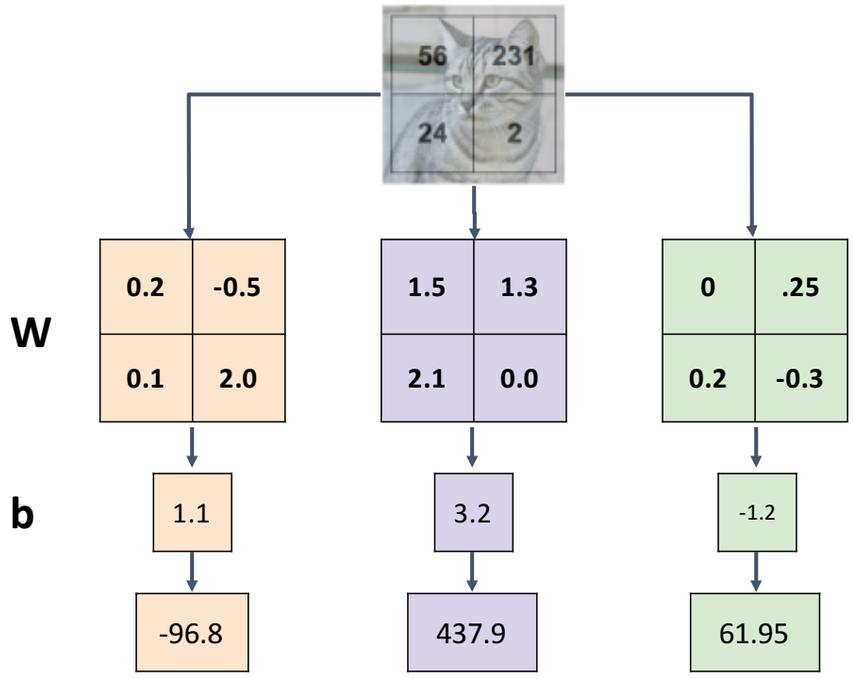
CIFAR-10 dataset



3-class linear classifier



Linear Classifier: Weights are Templates



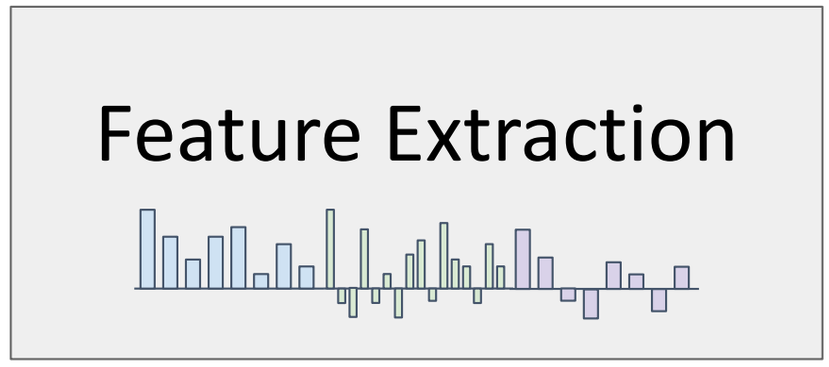
Linear classifier has one "template" per category

A single template cannot capture multiple modes of the data

e.g. horse template has 2 heads!

Feature Extraction for Linear Model

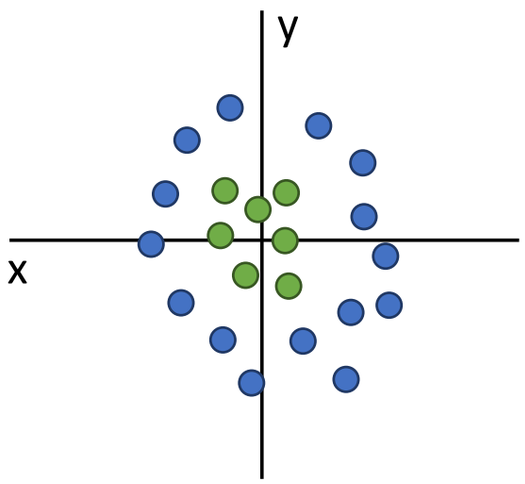
Image Features

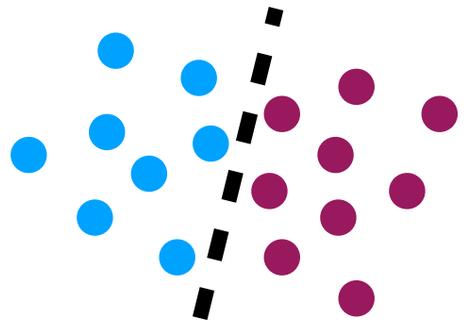


f

10 numbers giving scores for classes

training





Dedicated feature extraction is usually necessary for linear model learning to make the data more linear separable.

Example: Winner of 2011 ImageNet challenge

Low-level feature extraction ≈ 10k patches per image

- SIFT: 128-dim
- color: 96-dim

} reduced to 64-dim with PCA

FV extraction and compression:

- N=1,024 Gaussians, R=4 regions ⇨ 520K dim x 2
- compression: G=8, b=1 bit per dimension

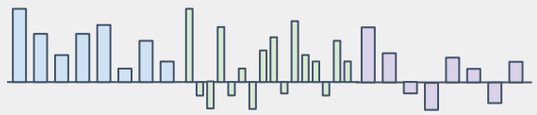
One-vs-all SVM learning with SGD

Late fusion of SIFT and color systems

F. Perronnin, J. Sánchez, "Compressed Fisher vectors for LSVRC", PASCAL VOC / ImageNet workshop, ICCV, 2011.

From Linear Model to Neural Network

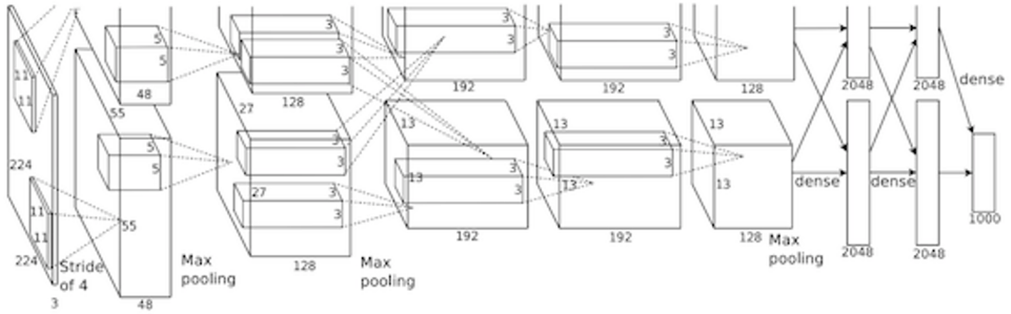
Feature Extraction



f

10 numbers giving scores for classes

training



Krizhevsky, Sutskever, and Hinton, "Imagenet classification with deep convolutional neural networks", NIPS 2012. Figure copyright Krizhevsky, Sutskever, and Hinton, 2012. Reproduced with permission.

10 numbers giving scores for classes

training

Neural networks have more complicated function mapping.
Handle complex data. Need less feature extraction.

Machine Learning: III

- From linear model to neural network
- Feedforward neural network
- Optimization
  - Stochastic gradient descent
  - Back-propagation
- Take-home messages

Feedforward Neural Network

Input: $x \in \mathbb{R}^D$ **Output**: $f(x) \in \mathbb{R}\text{^}C$

Before: Linear Classifier: $f(x) = Wx + b$
Learnable parameters: $W \in \mathbb{R}^{D \times C}, b \in \mathbb{R}^C$

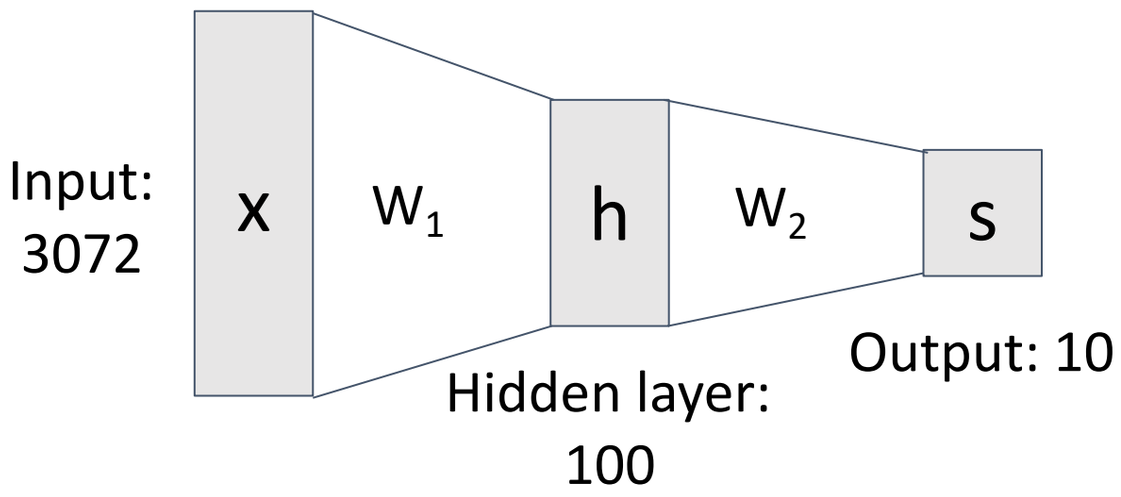
Feedforward Neural Network

Input: $x \in \mathbb{R}^D$ **Output**: $f(x) \in \mathbb{R}\text{^}C$

Before: Linear Classifier: $f(x) = Wx + b$
Learnable parameters: $W \in \mathbb{R}^{D \times C}, b \in \mathbb{R}^C$

Now: Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b2$

Learnable parameters: $W_1 \in \mathbb{R}^{H \times D}, b_1 \in \mathbb{R}^H, W_2 \in \mathbb{R}^{C \times H}, b_2 \in \mathbb{R}^C$



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Feedforward Neural Network

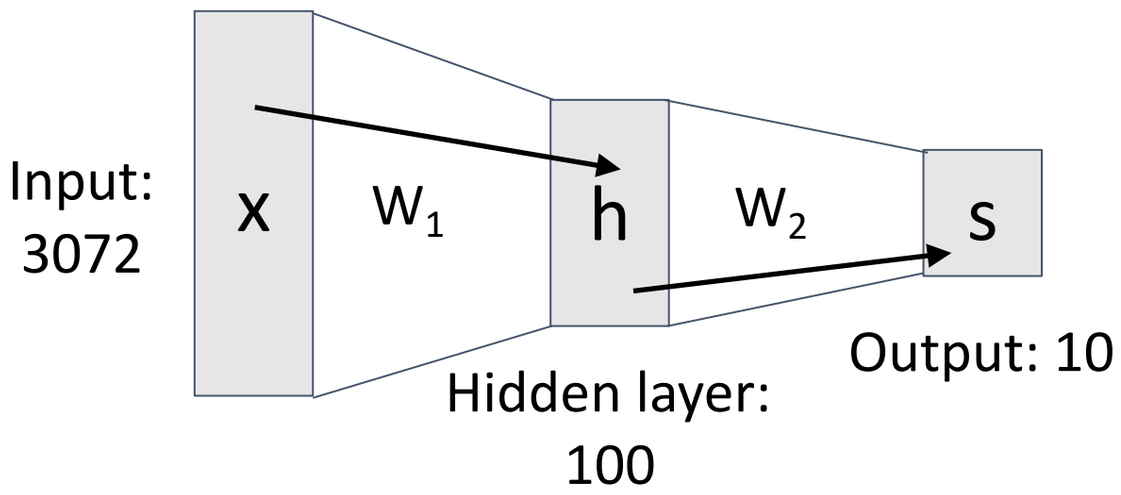
Input: $x \in \mathbb{R}^D$ **Output**: $f(x) \in \mathbb{R}\text{^}C$

Before: Linear Classifier: $f(x) = Wx + b$
Learnable parameters: $W \in \mathbb{R}^{D \times C}, b \in \mathbb{R}^C$

Now: Two-Layer Neural Network: $f(x) = W_2 \max(0, W_1 x + b_1) + b2$

Learnable parameters: $W_1 \in \mathbb{R}^{H \times D}, b_1 \in \mathbb{R}^H, W_2 \in \mathbb{R}^{C \times H}, b_2 \in \mathbb{R}^C$

Element (i, j) of W_1 gives the effect on h_i from x_j

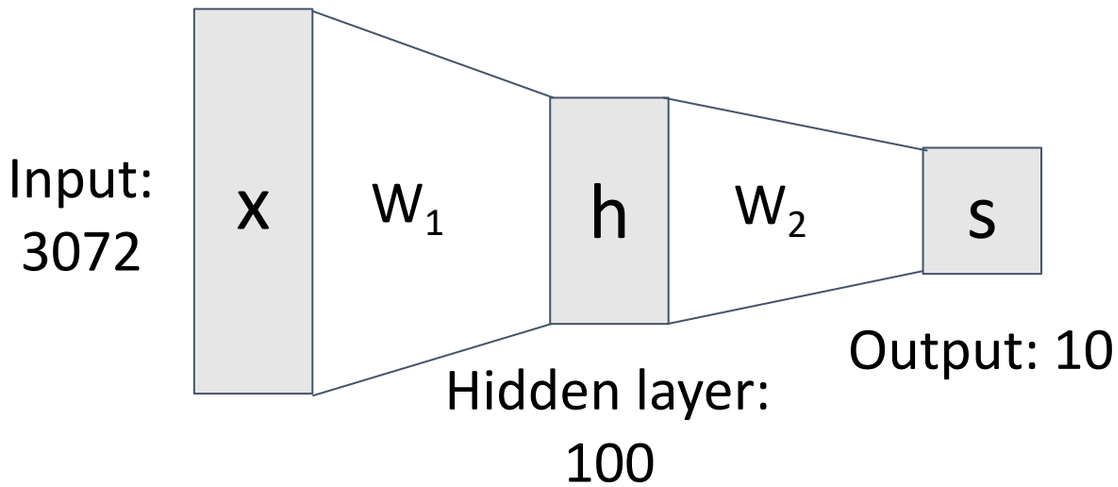


Element (i, j) of W_2 gives the effect on s_i from h_j

$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

Feedforward Neural Network

All elements of x affect all elements of h

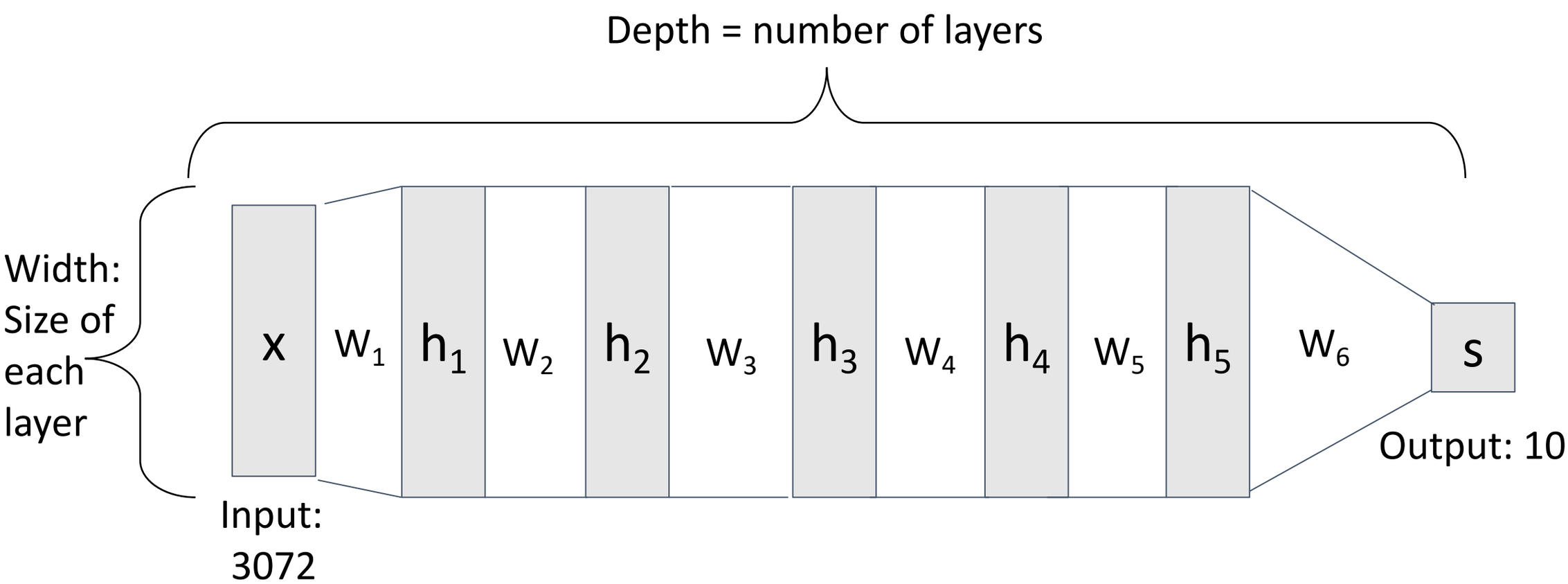


All elements of h affect all elements of s

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

Called Fully-connected feedforward neural network.

Neural Network with More Layers



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

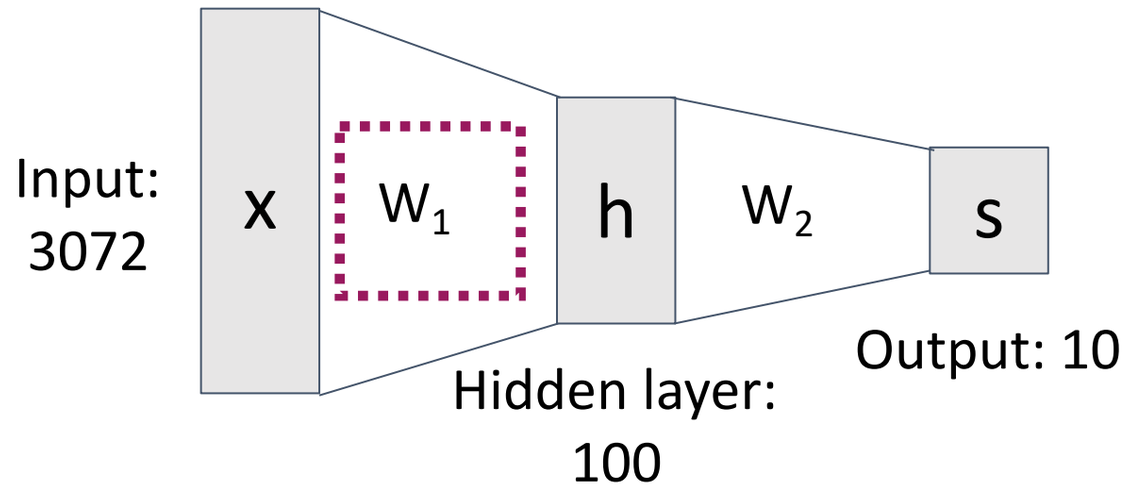
Fully-connected NNs with multiple hidden layers are usually called multi-layer perceptron (MLP).

Weight Template in NN

Can use different templates to cover multiple modes of a class!

(**Before**) Linear score function:

(**Now**) 2-layer Neural Network



$$x \in \mathbb{R}^D, W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}$$

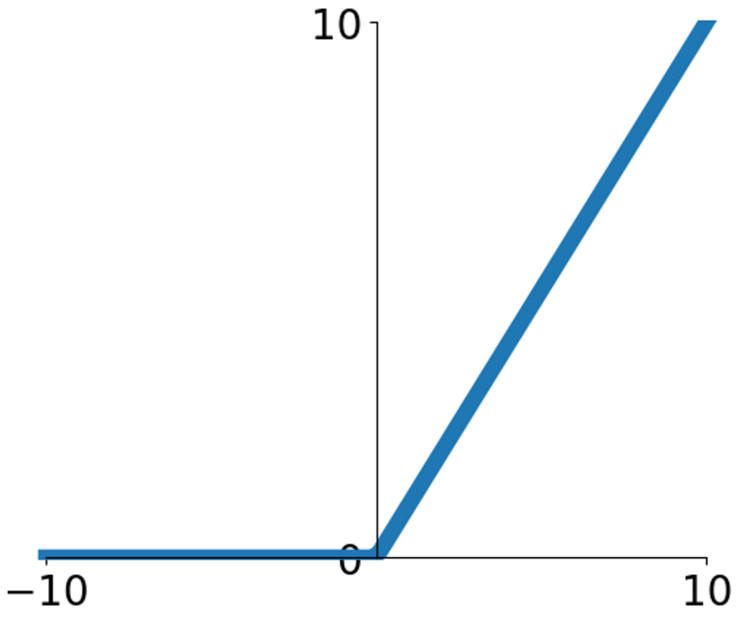
NN can learn more complicated feature templates, but may be more uninterpretable.

Activation Functions

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

The function $ReLU(z) = \max(0, z)$ is called “Rectified Linear Unit”

This is called the **activation function** of the neural network

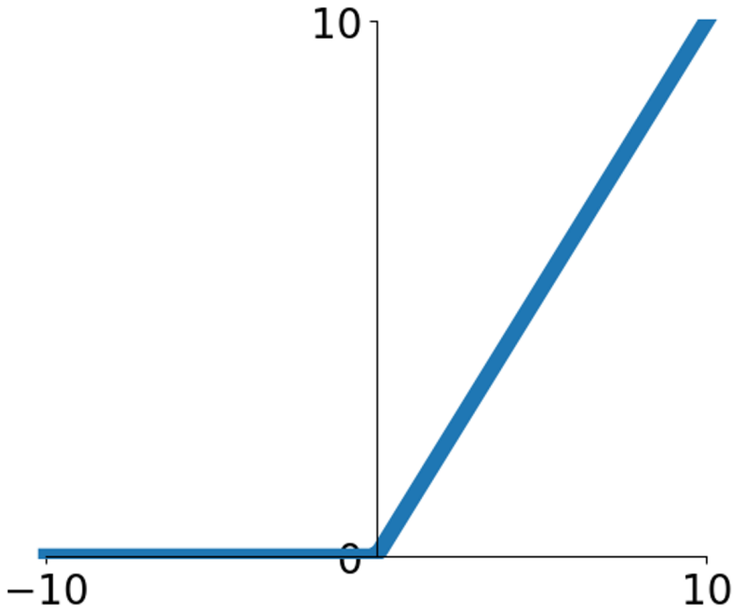


Activation Functions

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"

This is called the **activation function** of the neural network



Q: What happens if we build a neural network with no activation function?

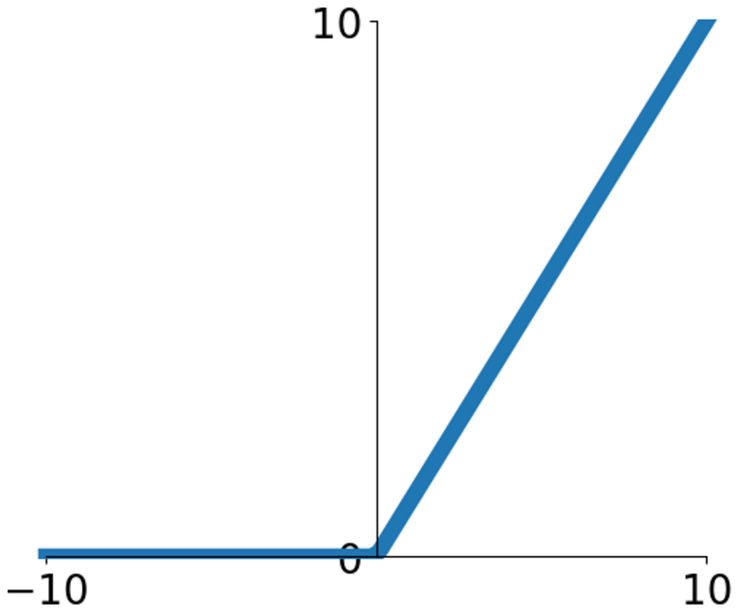
$$f(x) = W_2(W_1 x + b_1) + b_2$$

Activation Functions

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"

This is called the **activation function** of the neural network



Q: What happens if we build a neural network with no activation function?

$$f(x) = W_2(W_1 x + b_1) + b_2$$

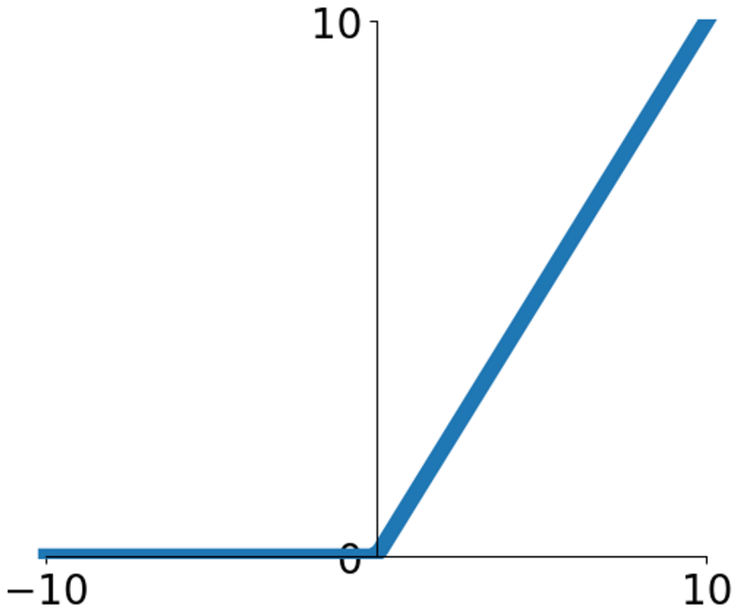
A: We end up with a linear classifier!

Activation Functions

2-layer Neural Network $f(x) = W_2 \max(0, W_1 x + b_1) + b_2$

The function $ReLU(z) = \max(0, z)$ is called "Rectified Linear Unit"

This is called the **activation function** of the neural network



Q: What happens if we build a neural network with no activation function?

$$f(x) = W_2(W_1 x + b_1) + b_2$$

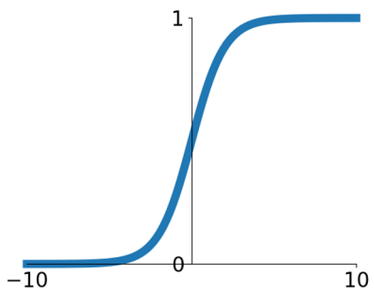
A: We end up with a linear classifier!

Activation functions are essential for obtaining non-linearity in NNs.

Activation Functions

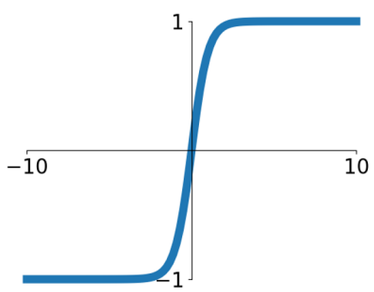
Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



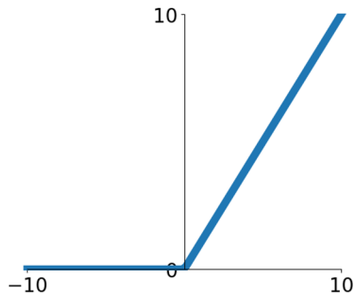
tanh

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



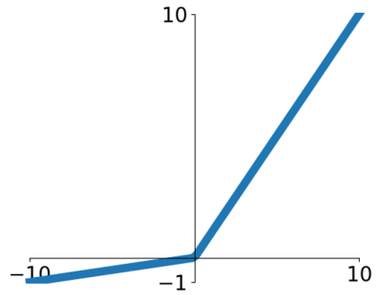
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.2x, x)$$

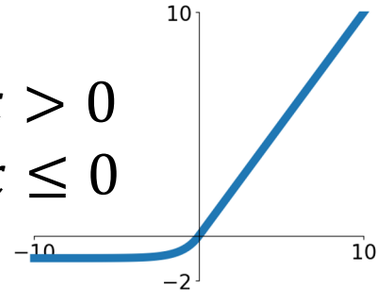


Softplus

$$\log(1 + \exp(x))$$

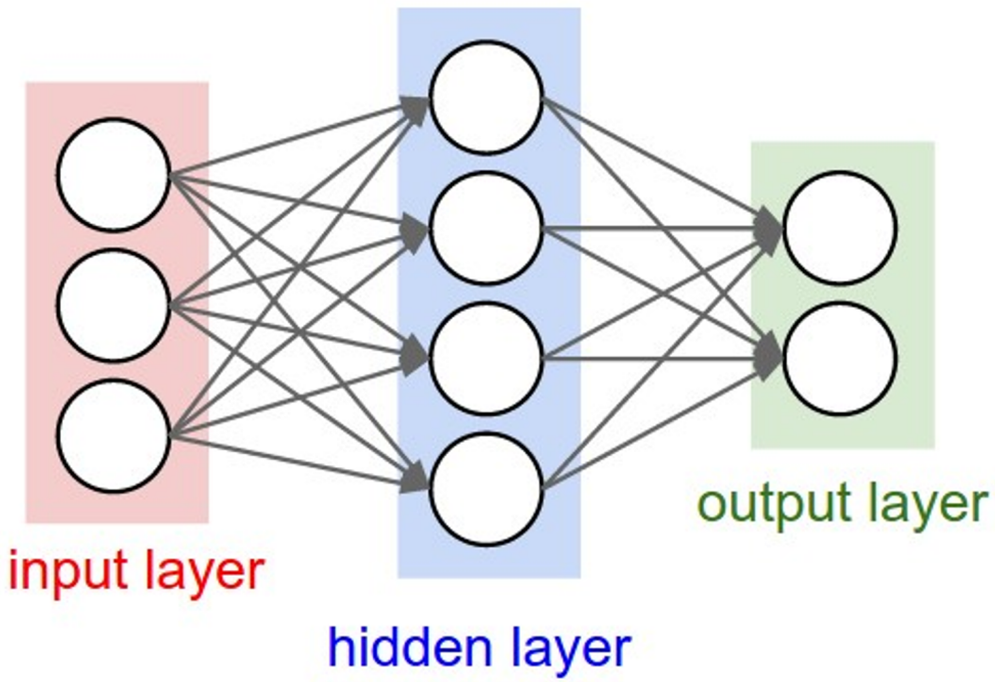
ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$



ReLU is the most common choice in today's NN learning.
Why use such simple activation?
We will explain in the next lecture.

Basic Implementation

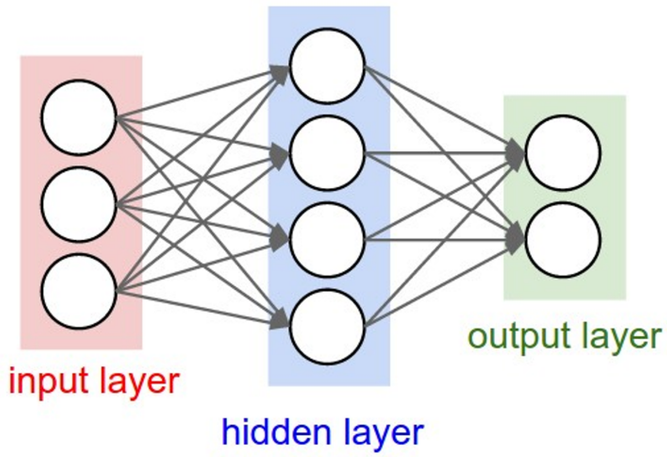



```python
import numpy as np
from numpy.random import randn

N, Din, H, Dout = 64, 1000, 100, 10
x, y = randn(N, Din), randn(N, Dout)
w1, w2 = randn(Din, H), randn(H, Dout)
for t in range(10000):
  h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  dy_pred = 2.0 * (y_pred - y)
  dw2 = h.T.dot(dy_pred)
  dh = dy_pred.dot(w2.T)
  dw1 = x.T.dot(dh * h * (1 - h))
  w1 -= 1e-4 * dw1
  w2 -= 1e-4 * dw2
```

Using only NumPy, you can implement this 2-layer NN with simple code.

Basic Implementation



Initialize weights and data

Compute loss (sigmoid activation, L2 loss)

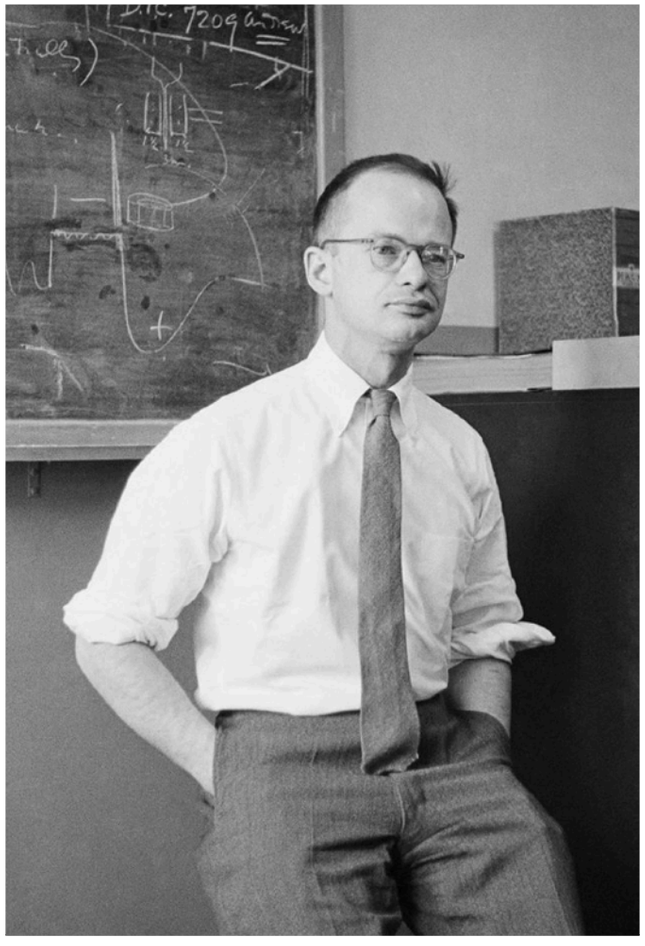
Compute gradients

introduce later

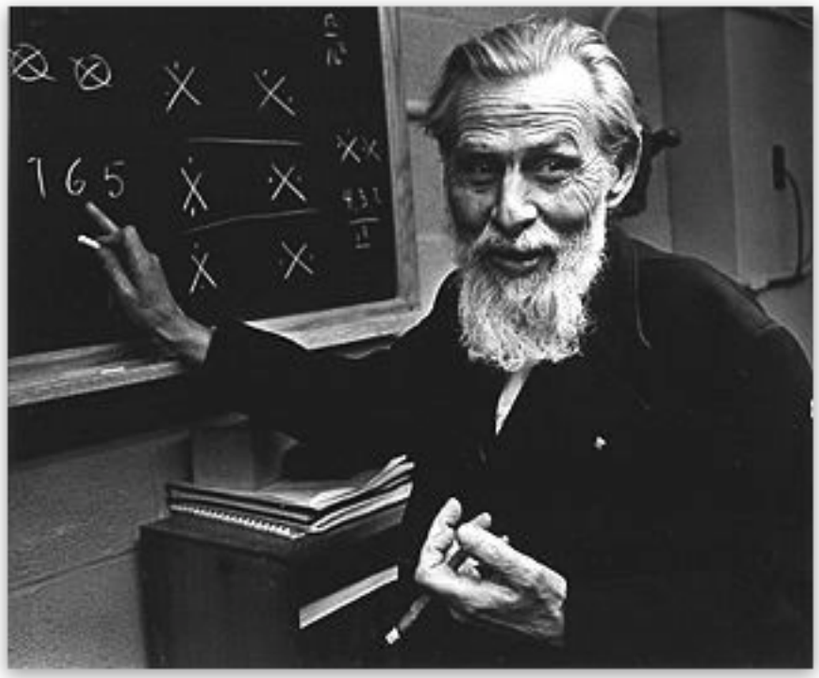
SGD step

```
1   import numpy as np
2   from numpy.random import randn
3
4   N, Din, H, Dout = 64, 1000, 100, 10
5   x, y = randn(N, Din), randn(N, Dout)
6   w1, w2 = randn(Din, H), randn(H, Dout)
7   for t in range(10000):
8     h = 1.0 / (1.0 + np.exp(-x.dot(w1)))
9     y_pred = h.dot(w2)
10    loss = np.square(y_pred - y).sum()
11    dy_pred = 2.0 * (y_pred - y)
12    dw2 = h.T.dot(dy_pred)
13    dh = dy_pred.dot(w2.T)
14    dw1 = x.T.dot(dh * h * (1 - h))
15    w1 -= 1e-4 * dw1
16    w2 -= 1e-4 * dw2
```

Why Called Neural Network?

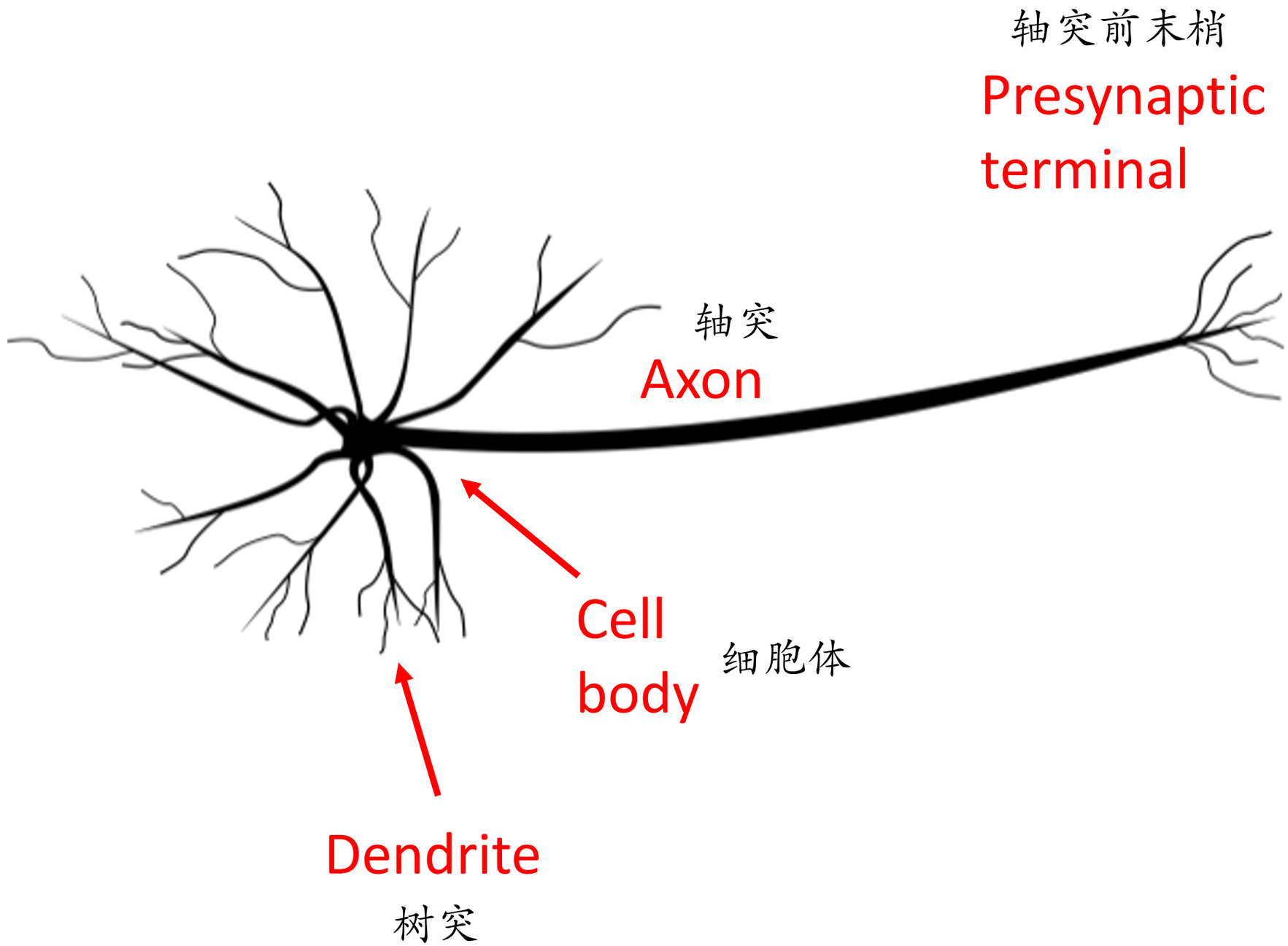


Walter Pitts

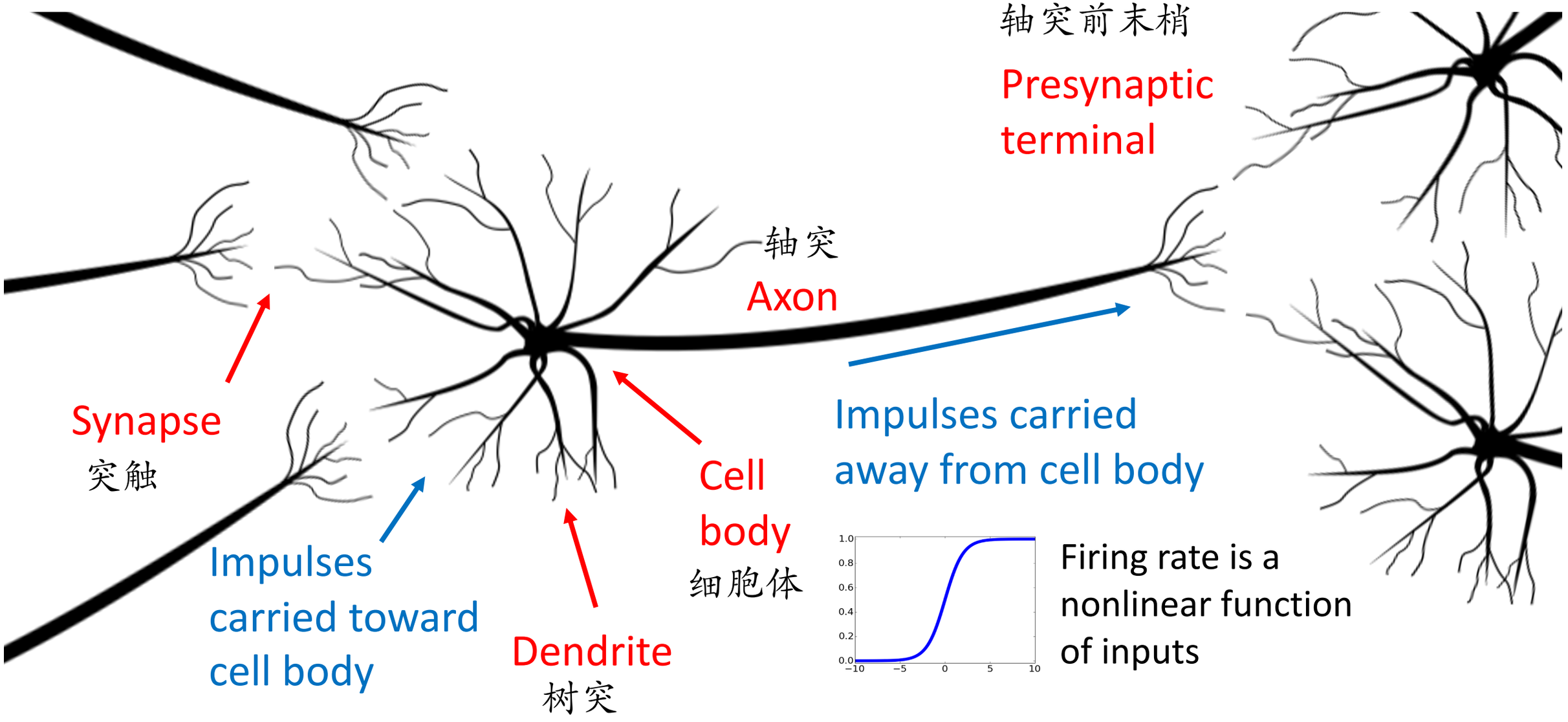


Warren MuCulloch

The Mechanism of Neurons



The Mechanism of Neurons

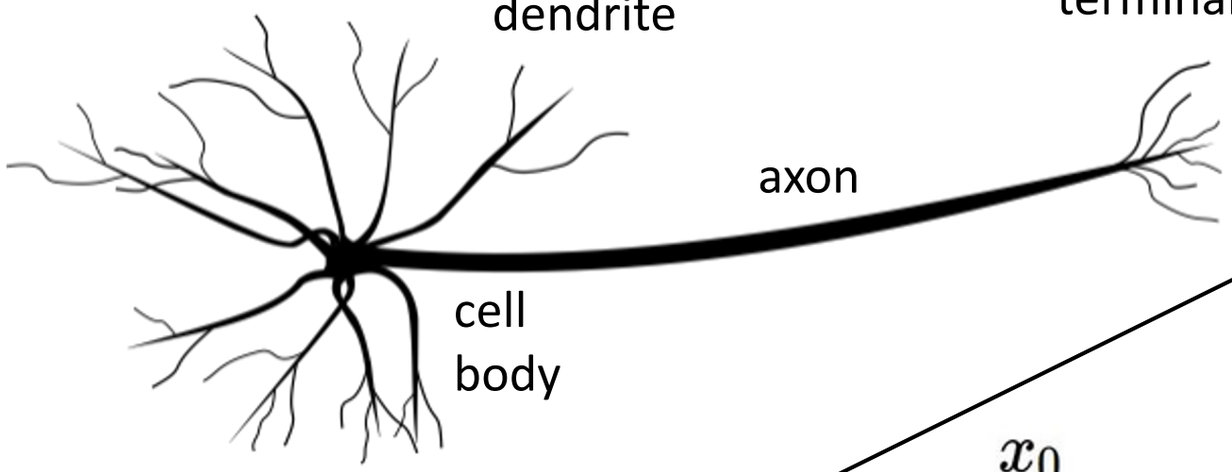


Firing rate is a nonlinear function of inputs

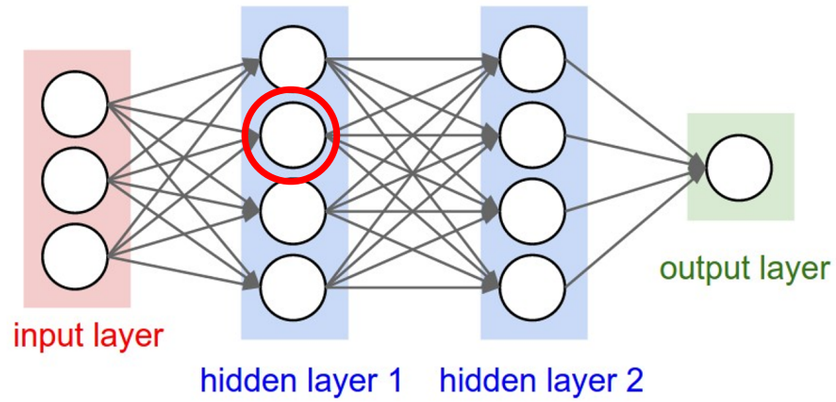
The neurons continuously receive and send signals to others.

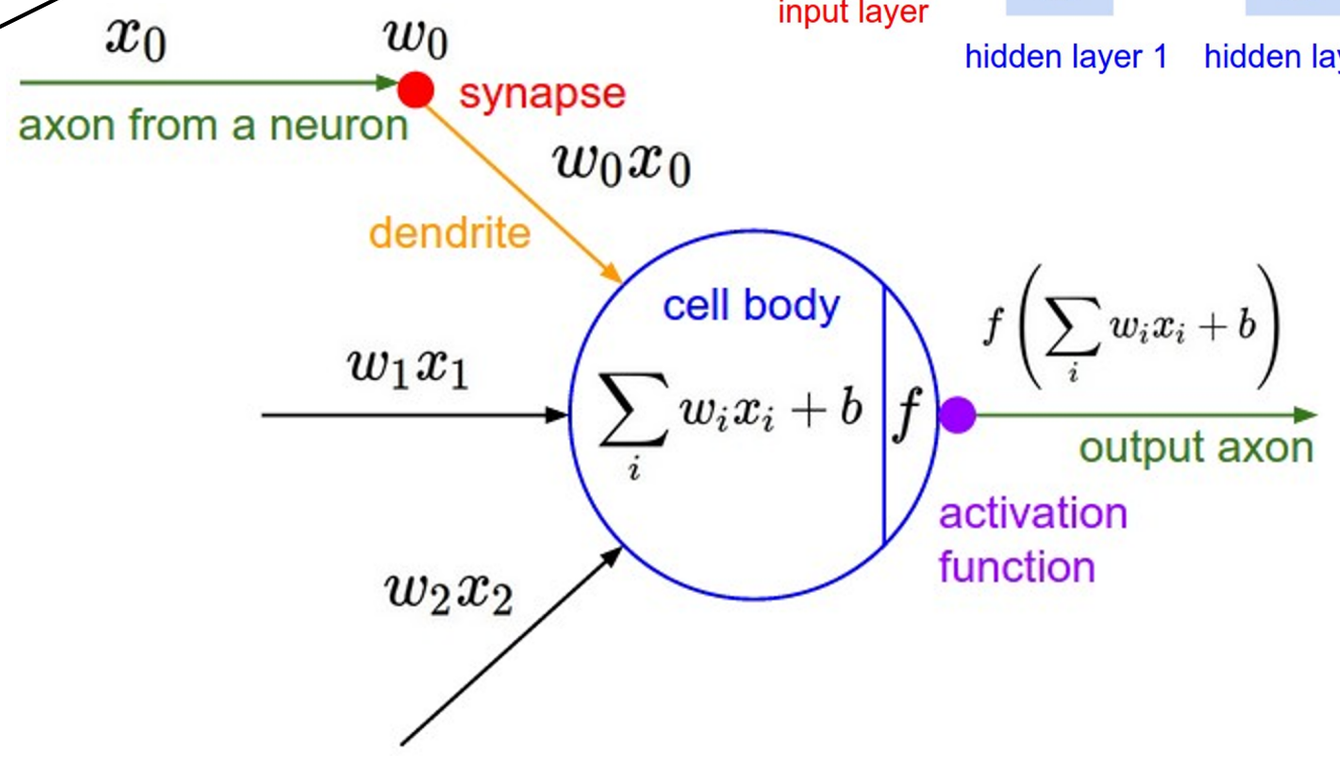
McCulloch-Pitts Neuron Model

Biological Neuron



Artificial Neuron

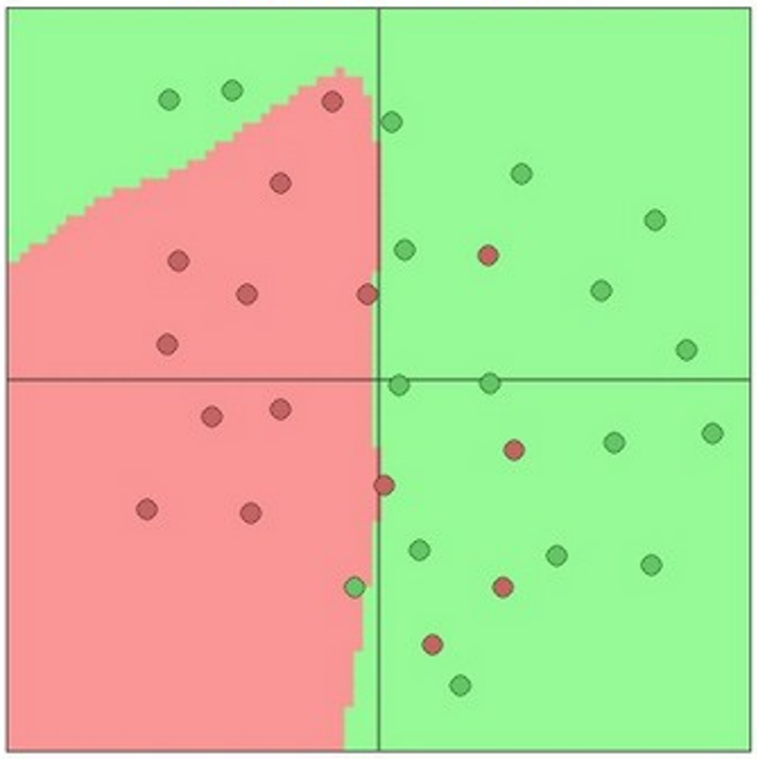




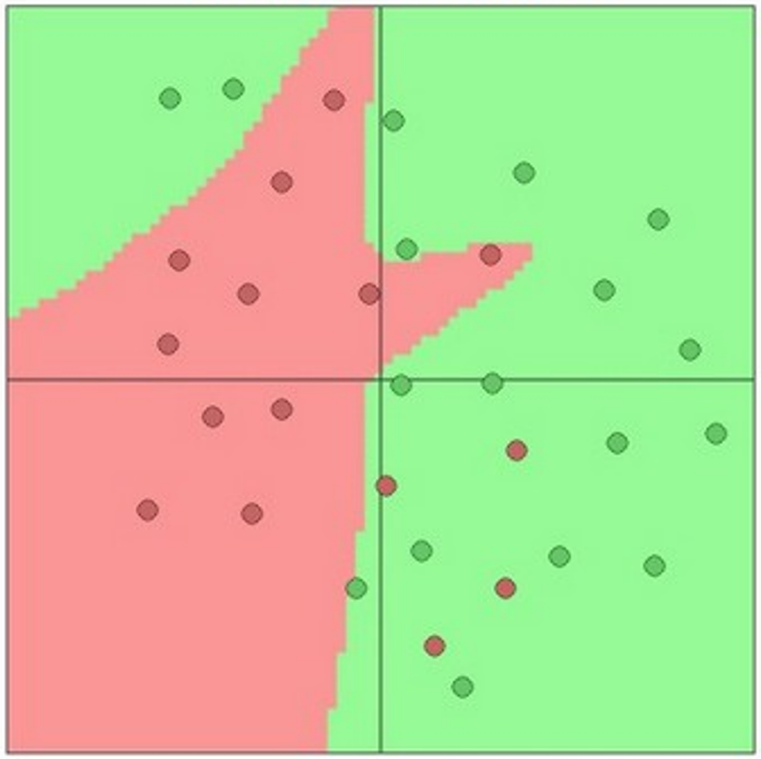
Only a strongly simplified model.
The real neuron works much more complicated than this.

Decision Boundary of Neural Network

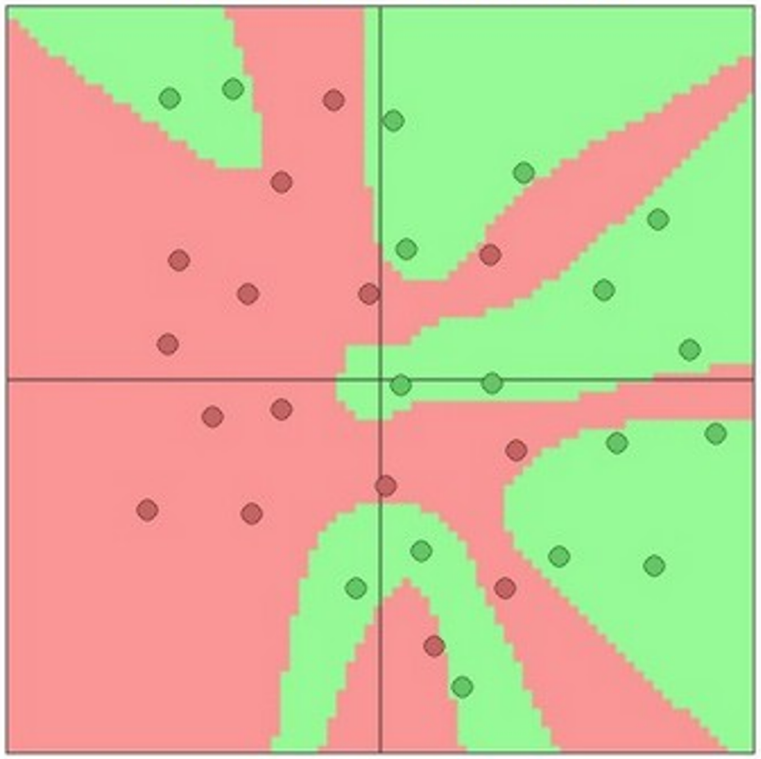
3 hidden units



6 hidden units



20 hidden units



More hidden units = more capacity

Universal Approximation of NN

Math. Control Signals Systems (1989) 2: 303–314

Mathematics of Control, Signals, and Systems

Approximation by Superpositions of a Sigmoidal Function*

G. Cybenko†

Abstract. In this paper we demonstrate that finite linear combinations of compositions of a fixed, univariate function and a set of affine functionals can uniformly approximate any continuous function of n real variables with support in the unit hypercube; only mild conditions are imposed on the univariate function. Our results settle an open question about representability in the class of single hidden layer neural networks. In particular, we show that arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity. The paper discusses approximation properties of other possible types of nonlinearities that might be implemented by artificial neural networks.

Key words. Neural networks, Approximation, Completeness.

Two-layer neural networks with sufficiently large width can approximate any continuous function.

Why go deep not wide? Still an open question.

How to optimize NN?

Machine Learning: III

- From linear model to neural network
- Feedforward neural network
- Optimization
  - Stochastic gradient descent
  - Back-propagation
- Take-home messages

Gradient Descent

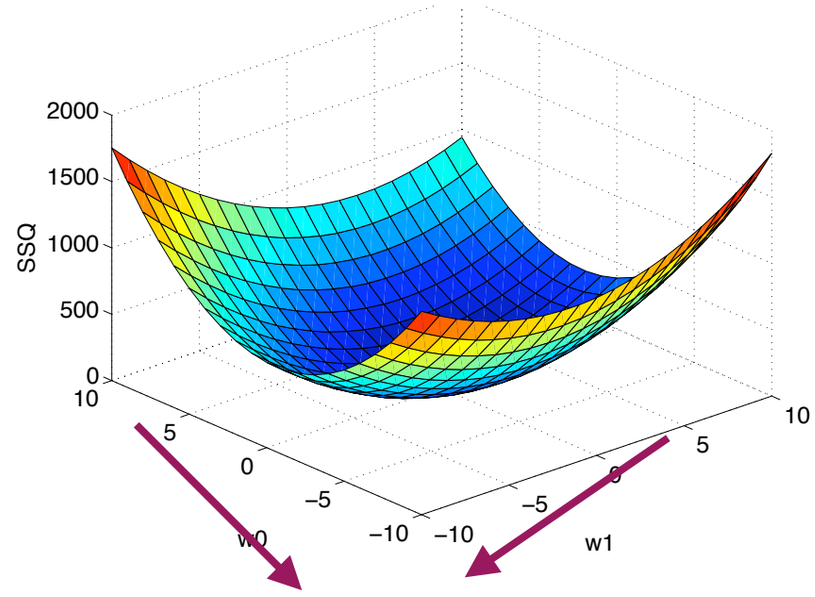
- Gradient descent is guaranteed to find the global minimum for convex functions.

Given an initial weight vector $\mathbf{w}_0$,

Do for $k=1, 2, \ldots$

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \, \partial Err(\mathbf{w}_k)/\partial \mathbf{w}_k$$

End when $|\mathbf{w}_{k+1} - \mathbf{w}_k| < \varepsilon$



Tracking the gradient direction to find the minimum

Gradient Descent

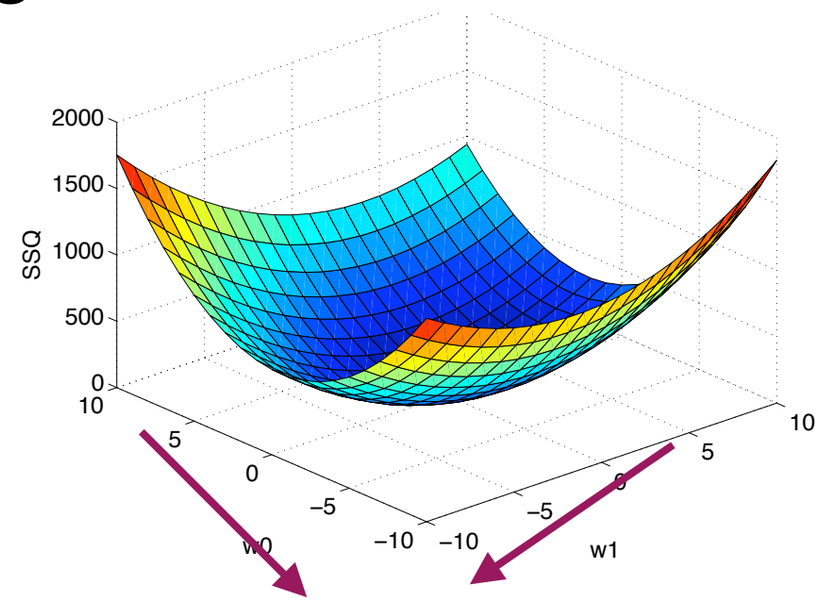
- Gradient descent is guaranteed to find the global minimum for convex functions.

Given an initial weight vector $\boldsymbol{w}_0$,

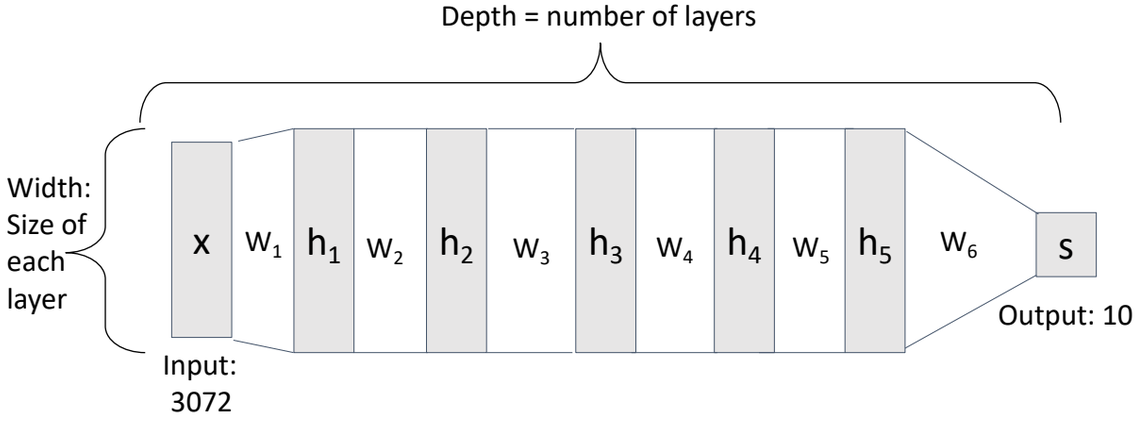
Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \, \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k| < \varepsilon$



Tracking the gradient direction to find the minimum



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

In general, NNs are non-convex functions. But fortunately, they are also suitable to use gradient descent to optimize.

SGD = Stochastic + Gradient

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \, \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1} - \boldsymbol{w}_k| < \varepsilon$

- Gradient descent uses all training data to calculate the loss.

SGD = Stochastic + Gradient

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

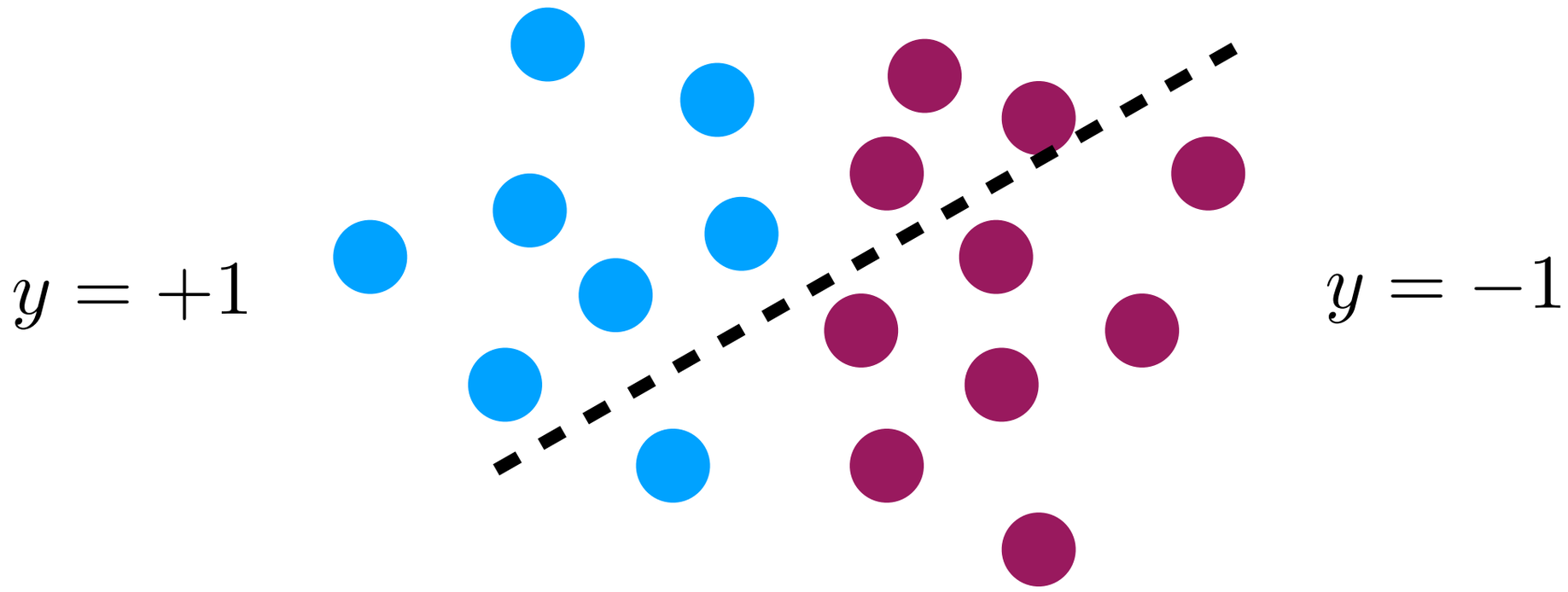
- Gradient descent uses all training data to calculate the loss.
- In each iteration, stochastic gradient descent (SGD) randomly samples a small subset of data to calculate the loss.

More computational efficient.
Lead to better solutions for NNs.

The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

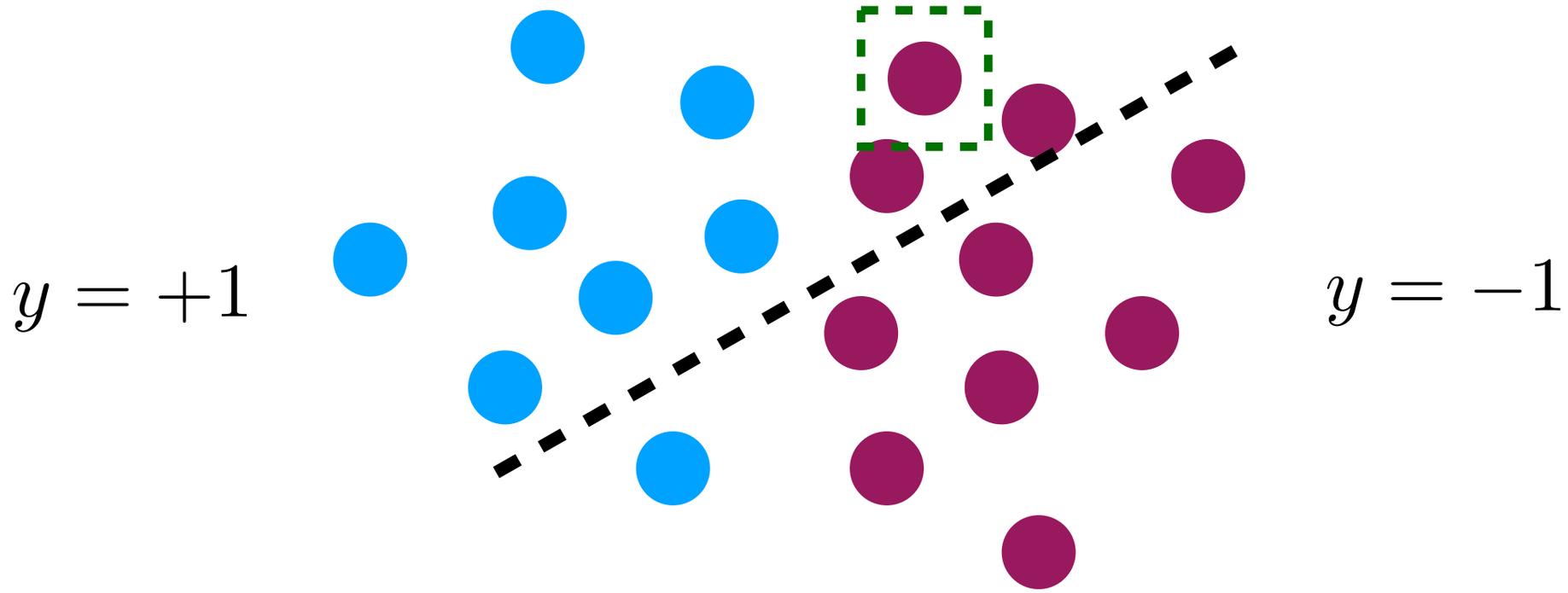
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

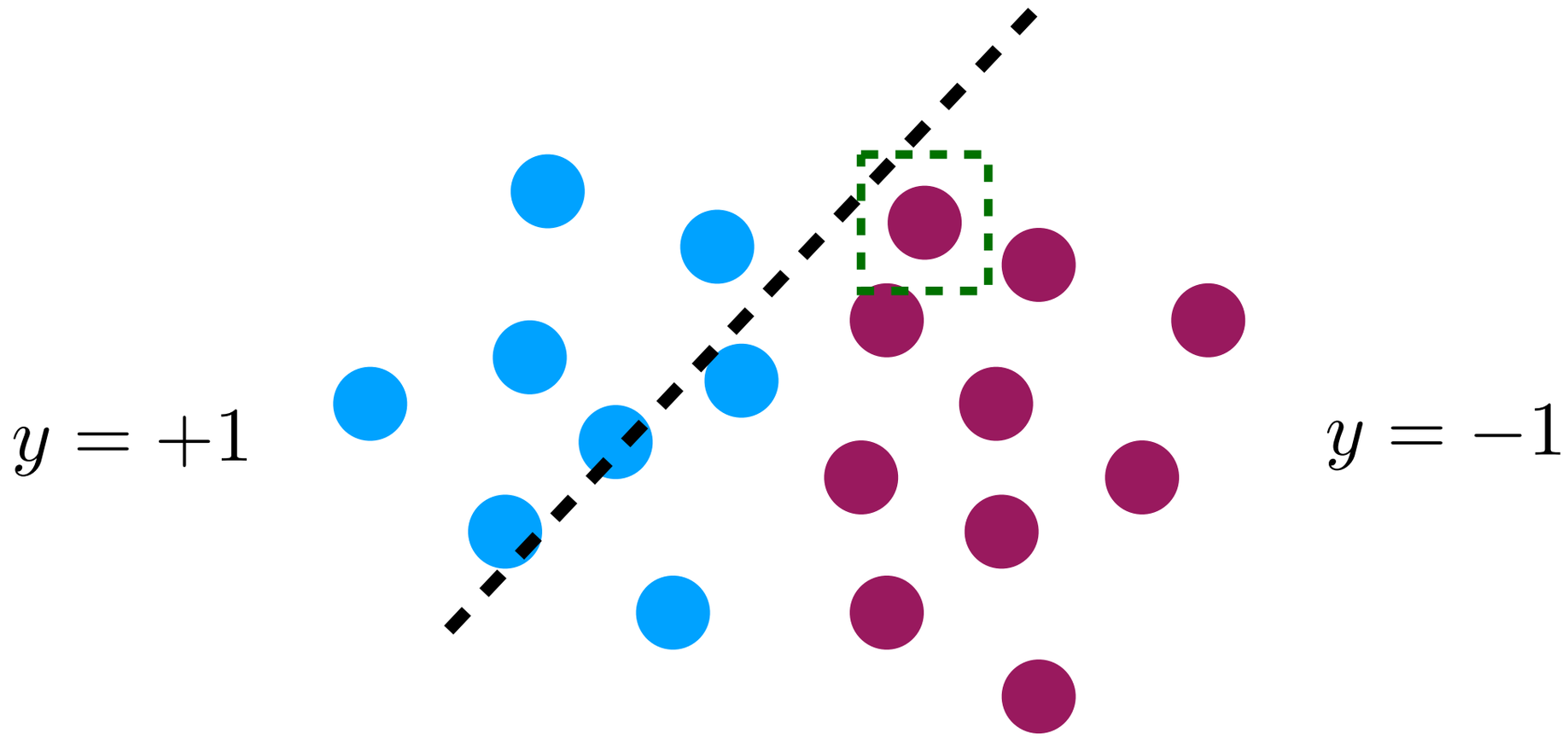
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

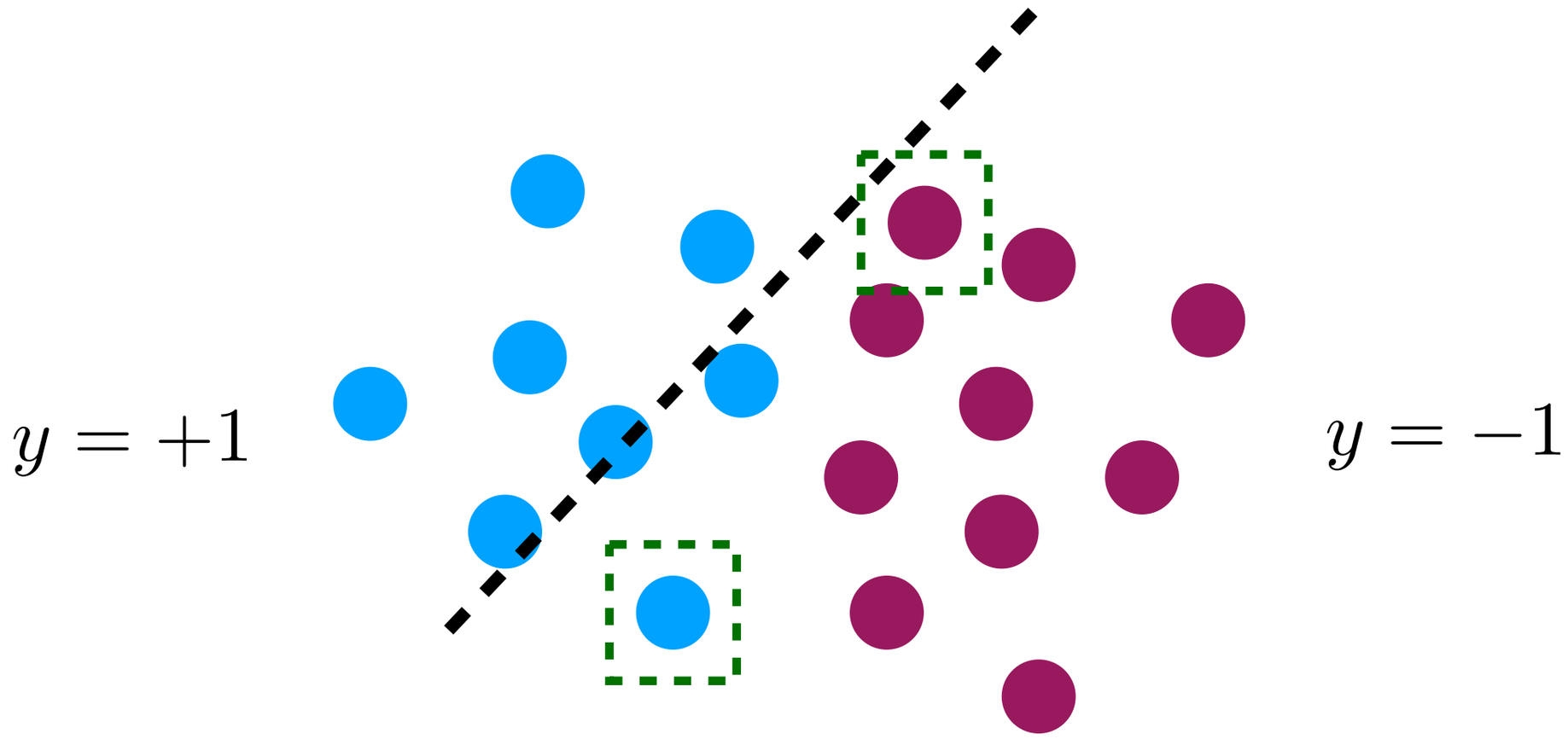
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

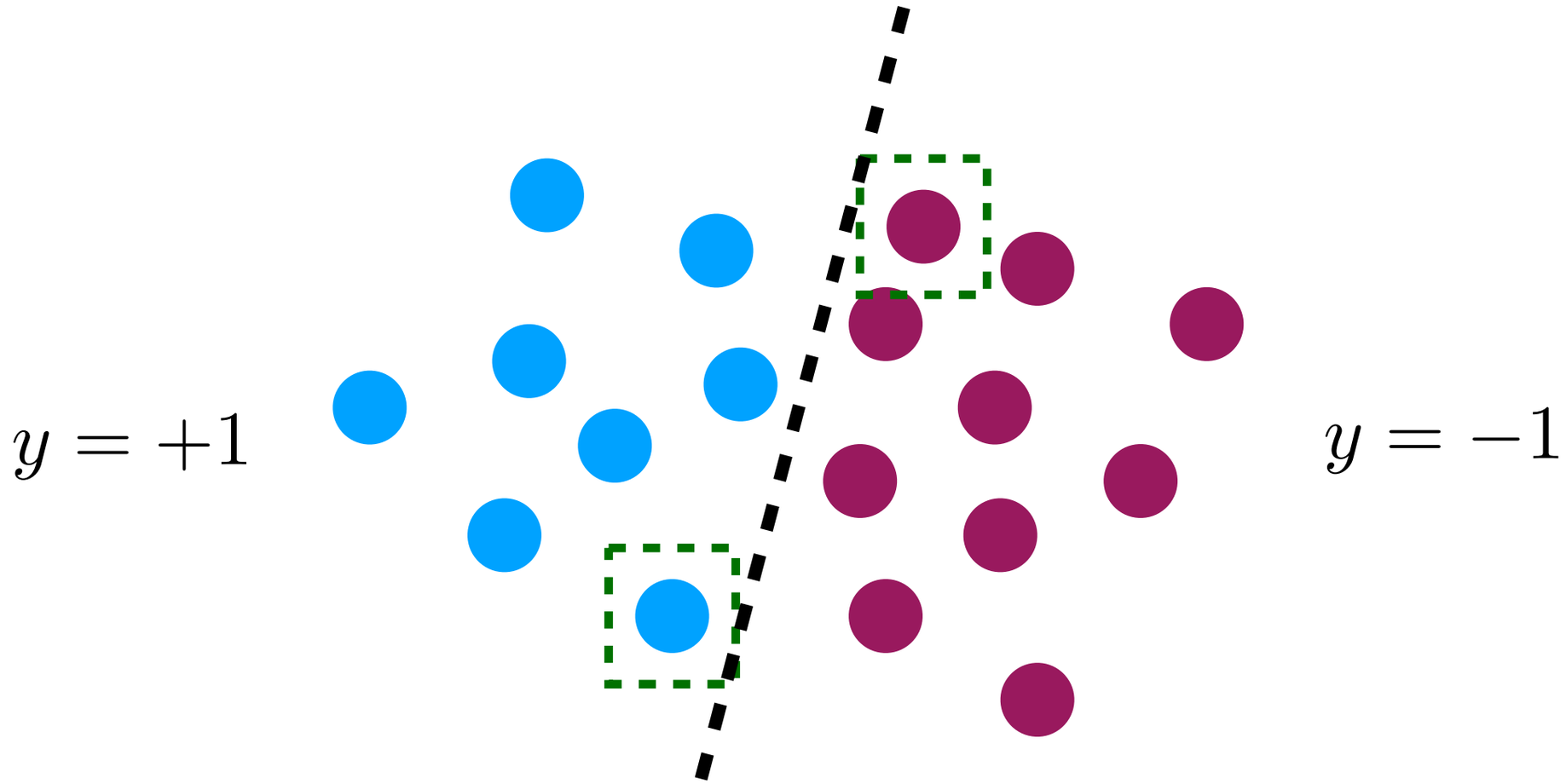
Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

Random sample instances and fix the hyperplane according to them.



The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

PERCEPTRON($\mathbf{w}_0$)

1 $\mathbf{w}_1 \leftarrow \mathbf{w}_0$ ▷ typically $\mathbf{w}_0 = \mathbf{0}$
2 **for** $t \leftarrow 1$ **to** T **do**
3 RECEIVE($\mathbf{x}_t$)
4 $\widehat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$
5 RECEIVE(y_t)
6 **if** $(\widehat{y}_t \neq y_t)$ **then**
7 $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$ ▷ more generally $\eta y_t \mathbf{x}_t, \eta > 0$.
8 **else** $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
9 **return** $\mathbf{w}_{T+1}$

The Perceptron Algorithm

- Perceptron is the prelude of SGD for linear model.

PERCEPTRON($\mathbf{w}_0$)

1 $\mathbf{w}_1 \leftarrow \mathbf{w}_0$ ▷ typically $\mathbf{w}_0 = \mathbf{0}$
2 **for** $t \leftarrow 1$ **to** T **do**
3 RECEIVE($\mathbf{x}_t$)
4 $\widehat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$
5 RECEIVE(y_t)
6 **if** $(\widehat{y}_t \neq y_t)$ **then**
7 $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$ ▷ more generally $\eta y_t \mathbf{x}_t, \eta > 0$.
8 **else** $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
9 **return** $\mathbf{w}_{T+1}$

The final hyperplane is the linear combination of sampled instances.
Similar to SVM, but SVM hyperplane restricts to the combination of support vectors.

SGD for Multi-Layer NN

- NNs are trained under various of objective functions. For regression, the common choice is MSE. For classification, the cross-entropy loss is often used:

$$-\sum_{i=1}^{N'}\Big[\sum_{c=1}^{C}\mathbb{I}[y_i = f(\mathbf{x}_i)]\log f(\mathbf{x}_i)\Big]$$

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k\, \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

SGD for Multi-Layer NN

- NNs are trained under various of objective functions. For regression, the common choice is MSE. For classification, the cross-entropy loss is often used:

$$-\sum_{i=1}^{N'}\Big[\sum_{c=1}^{C}\mathbb{I}[y_i = f(\mathbf{x}_i)]\log f(\mathbf{x}_i)\Big]$$

SGD: sample a small subset of data to calculate

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

SGD for Multi-Layer NN

- NNs are trained under various of objective functions. For regression, the common choice is MSE. For classification, the cross-entropy loss is often used:

$$-\sum_{i=1}^{N'}\Big[\sum_{c=1}^{C}\mathbb{I}[y_i = f(\mathbf{x}_i)]\log f(\mathbf{x}_i)\Big]$$

SGD: sample a small subset of data to calculate

Given an initial weight vector $\boldsymbol{w}_0$,

Do for $k=1, 2, \ldots$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k \partial Err(\boldsymbol{w}_k)/\partial \boldsymbol{w}_k$$

End when $|\boldsymbol{w}_{k+1}-\boldsymbol{w}_k| < \varepsilon$

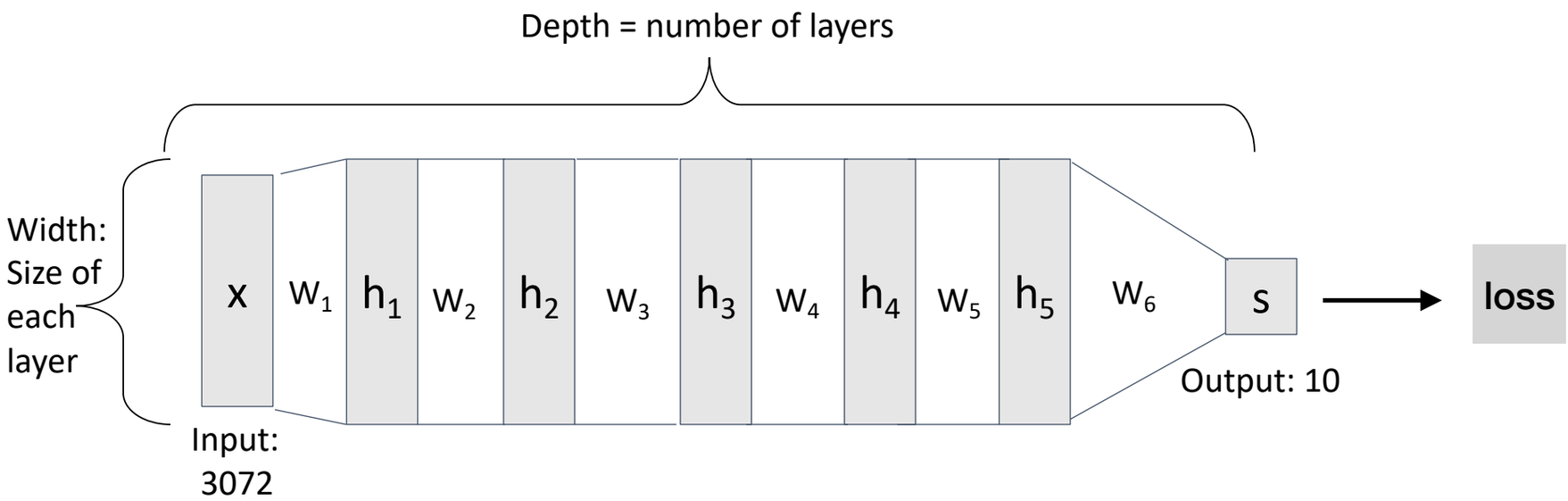
Parameter of NNs are the weights for all layers.
The key is to obtain their gradients efficiently.

Machine Learning: III

- From linear model to neural network
- Feedforward neural network
- Optimization
  - Stochastic gradient descent
  - Back-propagation
- Take-home messages

Back Propagation vs. Compositional Derivation

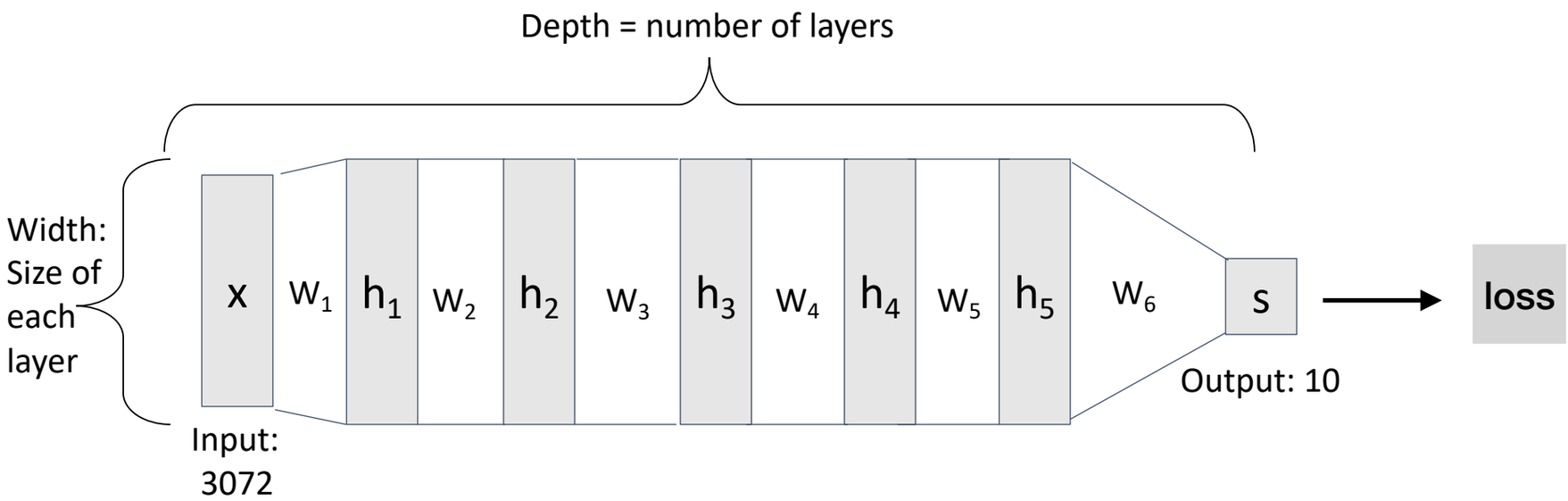
- Neural networks are compositional functions with multiple layers.



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

Back Propagation vs. Compositional Derivation

- Neural networks are compositional functions with multiple layers.

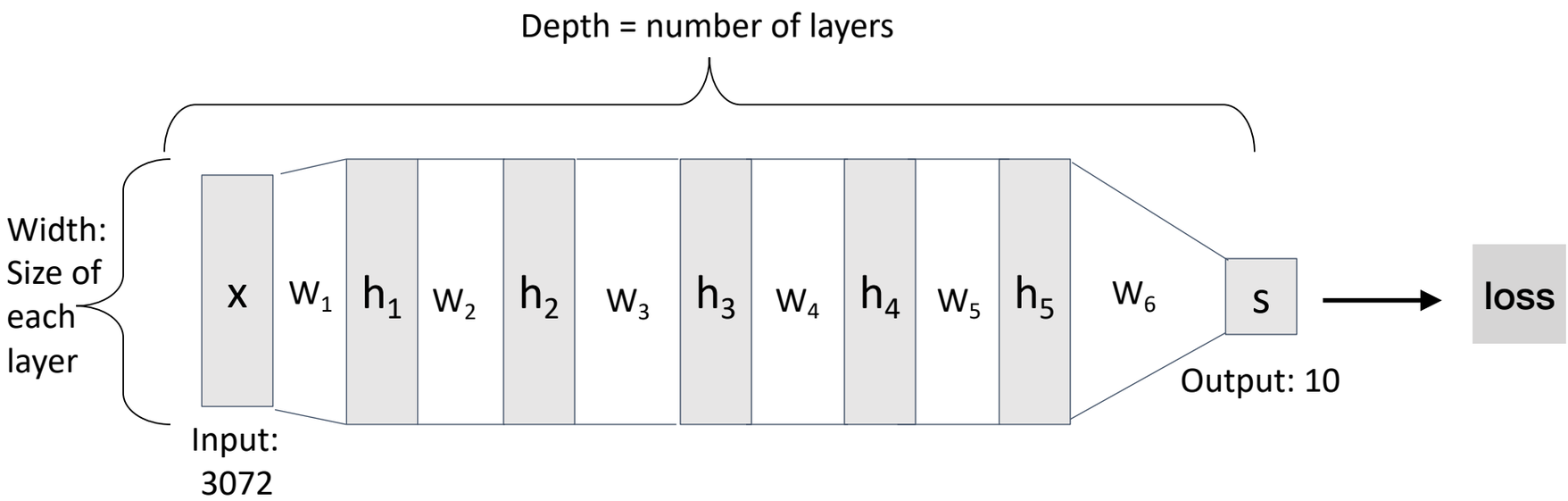


$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

- In fact, calculating the gradient of the parameters are just compositional derivation. Why do we need a new algorithm?

Back Propagation vs. Compositional Derivation

- Neural networks are compositional functions with multiple layers.



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

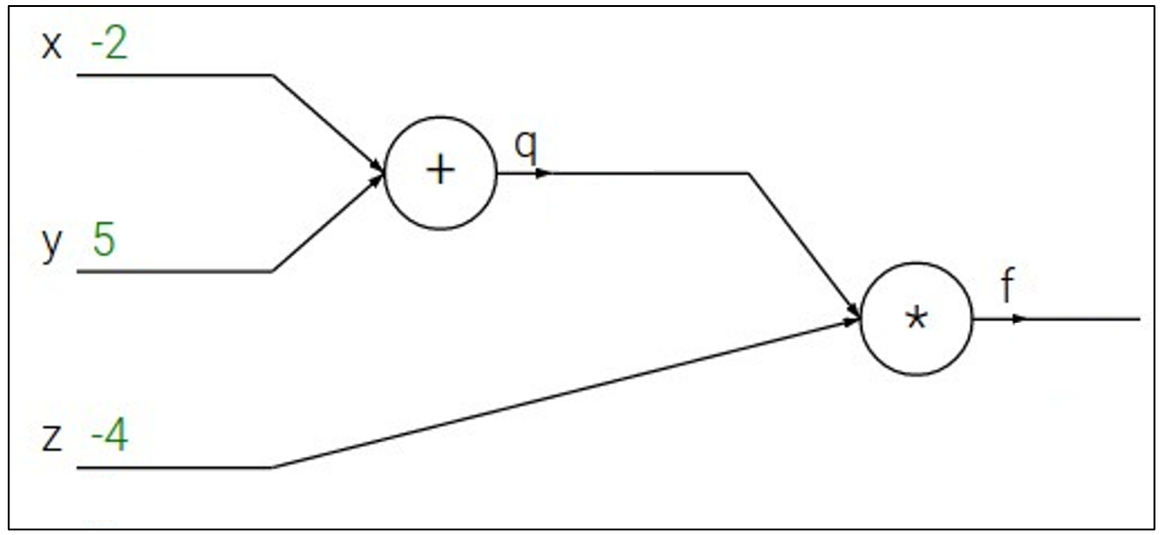
- In fact, calculating the gradient of the parameters are just compositional derivation. Why do we need a new algorithm?

We should do this more efficiently!

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4

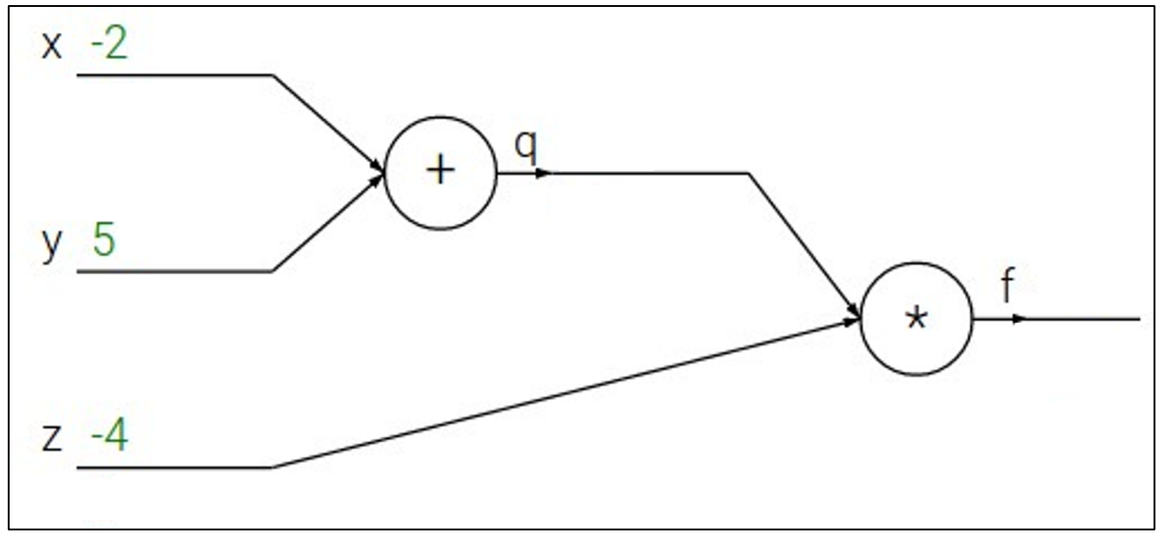


- Want to compute the gradient value of x, y, z given the inputs.

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



- Want to compute the gradient value of x, y, z given the inputs.

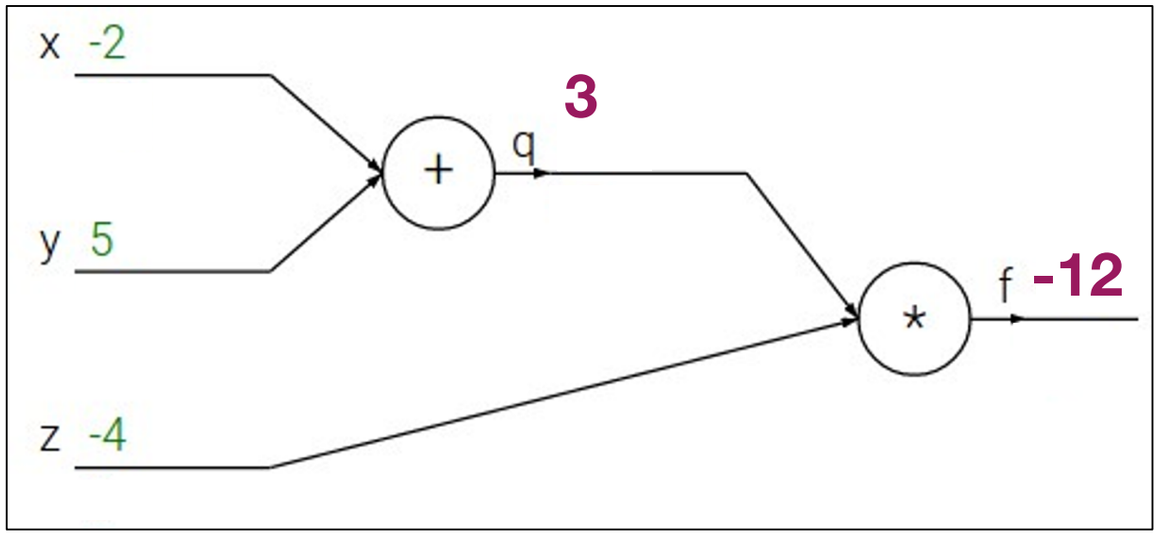
1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



- Want to compute the gradient value of x, y, z given the inputs.

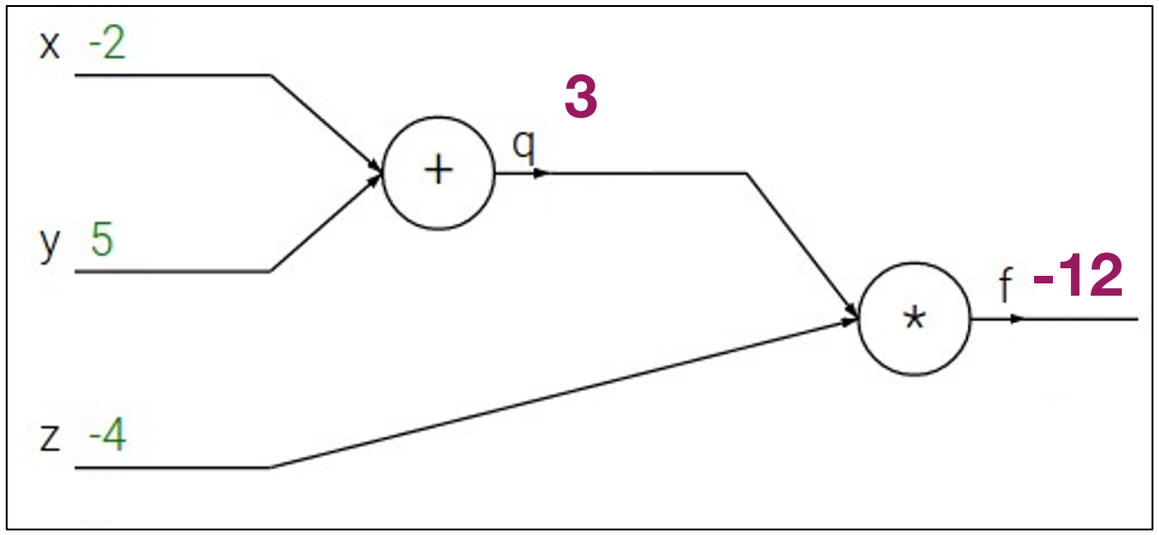
1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

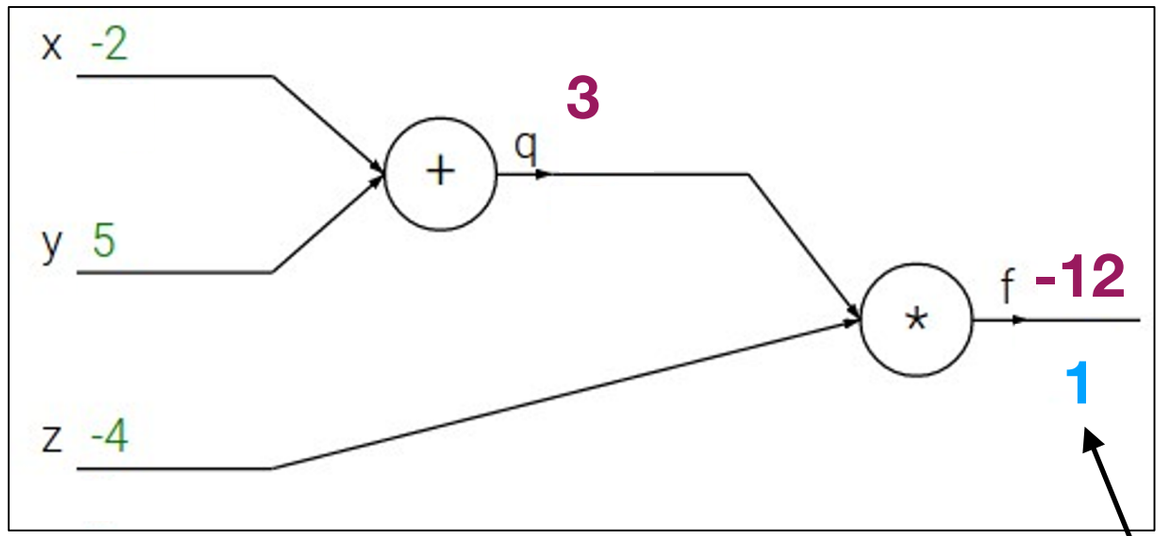
2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



- Want to compute the gradient value of x, y, z given the inputs. $\frac{\partial f}{\partial f}$

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

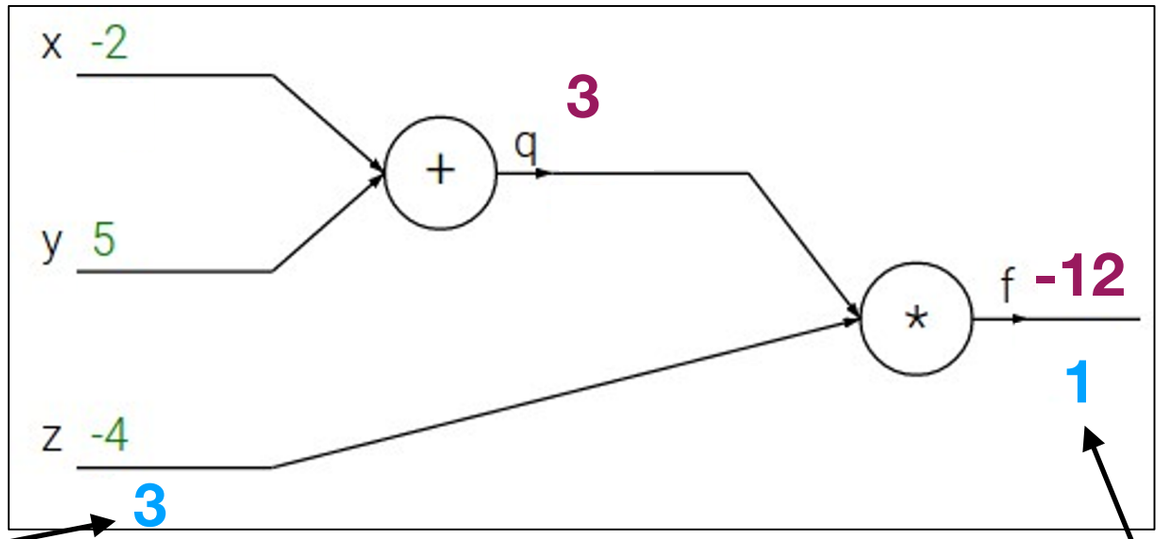
2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial f}$$

- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

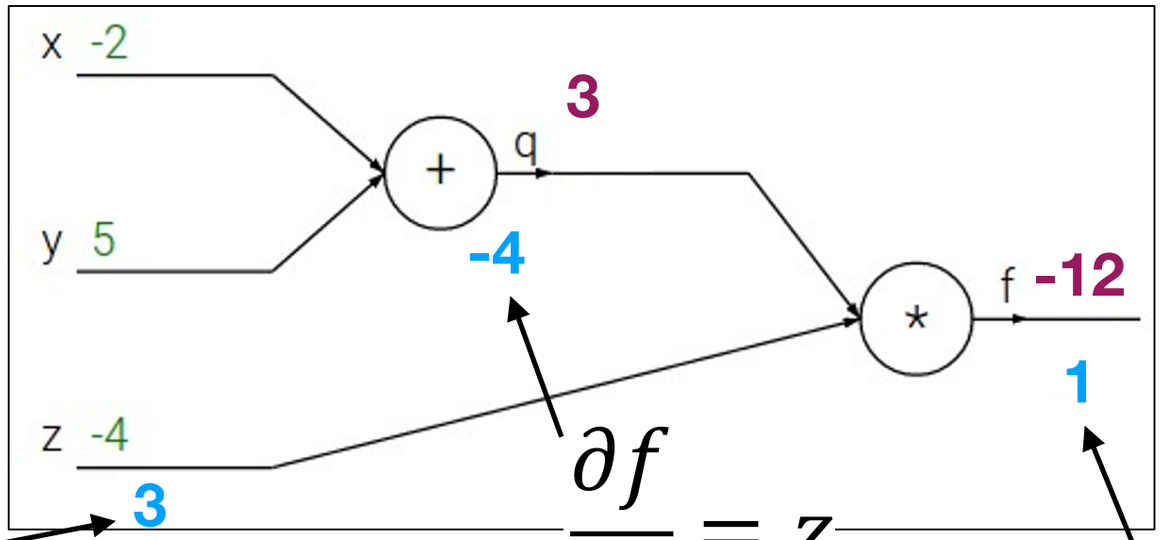
2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Back Propagation

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z \qquad \frac{\partial f}{\partial f}$$

- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

$$q = x + y \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

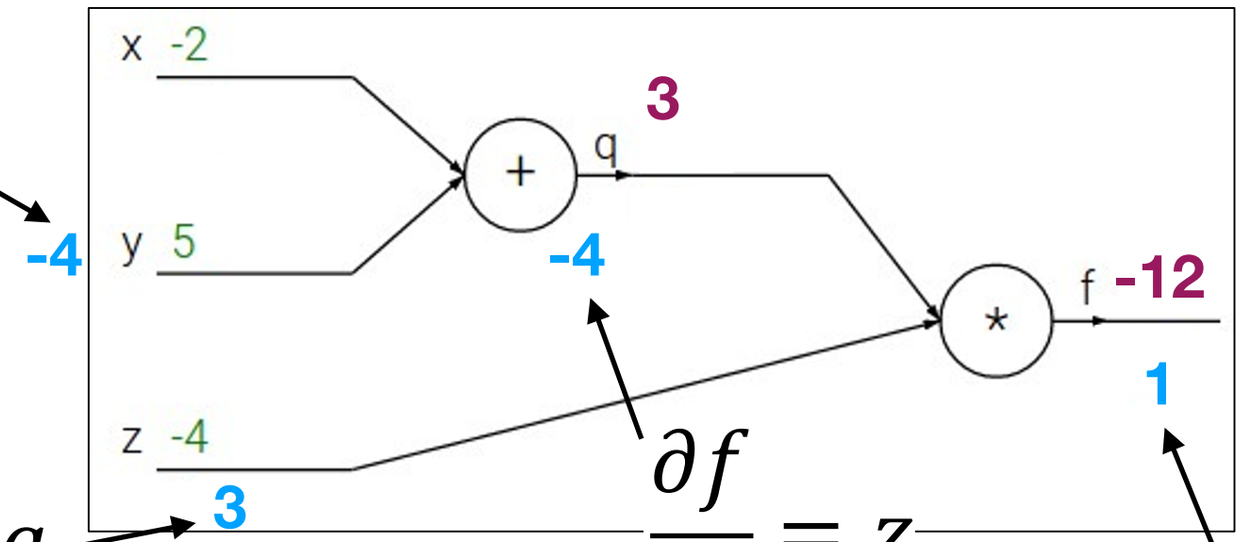
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Back Propagation

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial f}{\partial f}$$

- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

$$q = x + y \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

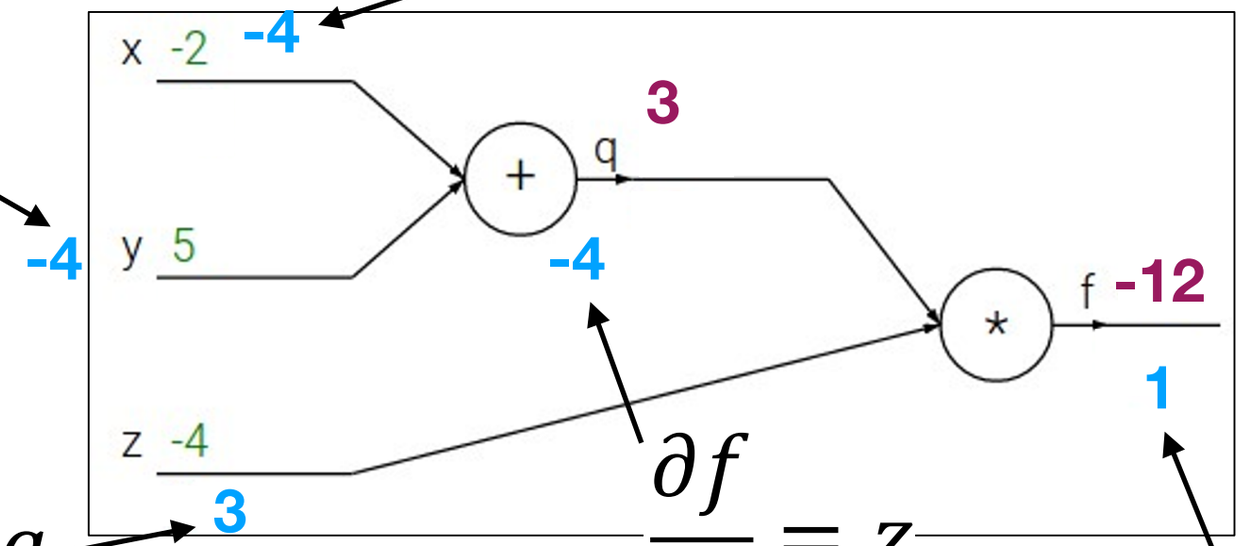
Back Propagation

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial q} = z$$

$$\frac{\partial f}{\partial f}$$

- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

$$q = x + y \quad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

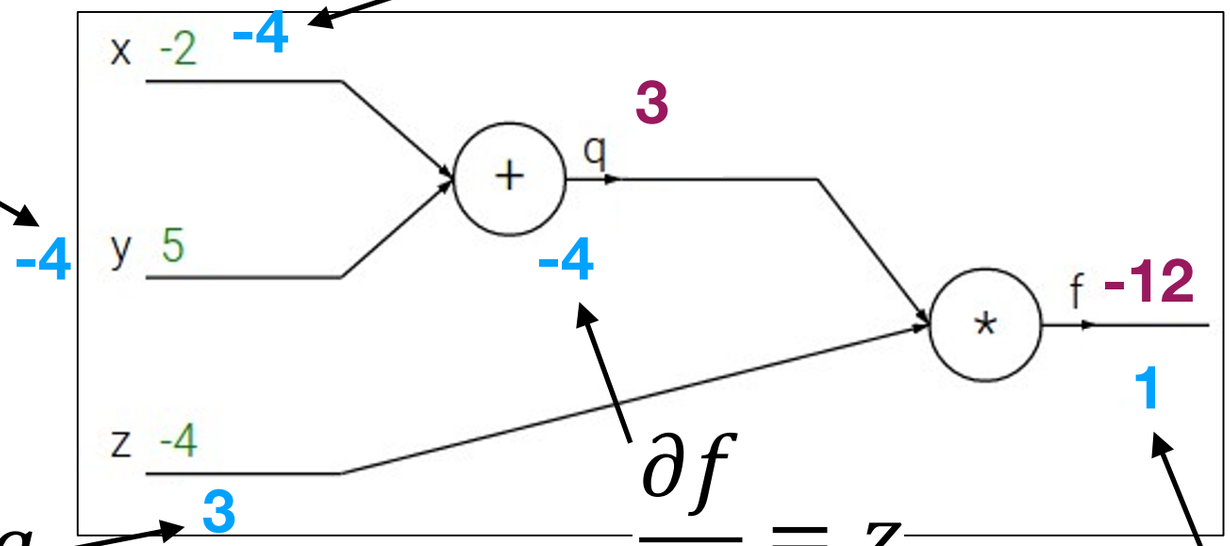
Back Propagation

$$\frac{\partial f}{\partial y} = \frac{\partial q}{\partial y}\frac{\partial f}{\partial q}$$

$$\frac{\partial f}{\partial x} = \frac{\partial q}{\partial x}\frac{\partial f}{\partial q}$$

$$f(x, y, z) = (x + y) \cdot z$$

e.g. x = -2, y = 5, z = -4



$$\frac{\partial f}{\partial z} = q$$

$$\frac{\partial f}{\partial q} = z$$

$$\frac{\partial f}{\partial f}$$

- Want to compute the gradient value of x, y, z given the inputs.

1. Forward pass: Compute outputs

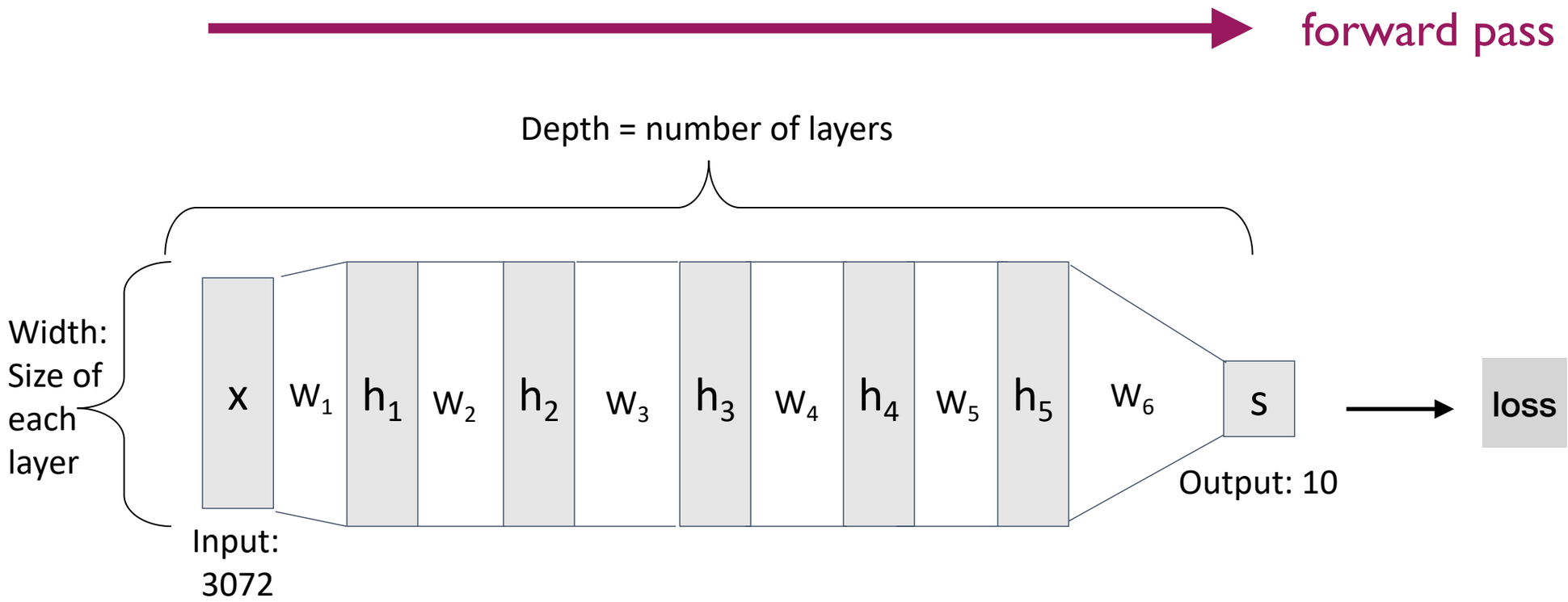
$$q = x + y \qquad f = q \cdot z$$

2. Backward pass: Compute derivatives

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

The key is to store the intermediate values during forward and backward passes.
The similar idea of dynamic programming!

Back Propagation



$$s = W_6 \max(0, W_5 \max(0, W_4 \max(0, W_3 \max(0, W_2 \max(0, W_1 x)))))$$

backward pass

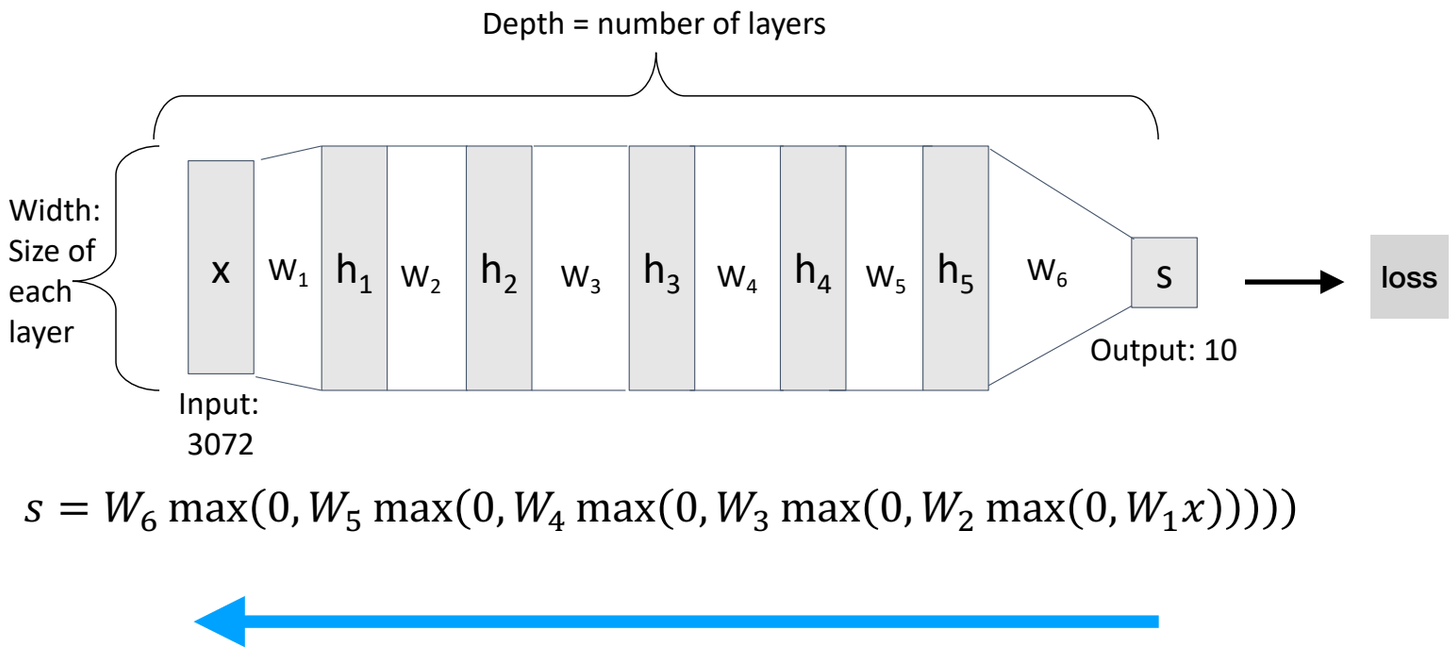
NNs can be formulated into computational graphs, in which nodes represent values and parameters, edges represent operators.
Both training and prediction can be realized by forward and backward passes through the computational graphs.

Deep Learning Tools

TensorFlow

PyTorch

飞桨 PaddlePaddle



Deep learning tools are designed to efficiently manage the computational graphs, In special tensor calculation and automatic derivation.

Machine Learning: III

- From linear model to neural network
- Feedforward neural network
- Optimization
  - Stochastic gradient descent
  - Back-propagation
- Take-home messages

Take-Home Messages

- Logistic regression is the linear model optimizing logistic loss, which is the special case of the cross-entropy loss.

- Feedforward neural networks are generalizations of the linear model: multiple layers of linear transformations + activations.

- Backward propagation (BP) is to do stochastic gradient descent optimization.

- BP = compositional derivation + dynamic programming.

Thanks for your attention!
Discussions?

Acknowledgement: Many materials in this lecture are taken from Justin Johnson: https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/schedule.html